

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 10-078880

(43)Date of publication of application : 24.03.1998

(51)Int.Cl.

G06F 9/46

G06F 9/38

G06F 15/16

(21)Application number : 08-249272

(71)Applicant : NEC CORP

(22)Date of filing : 30.08.1996

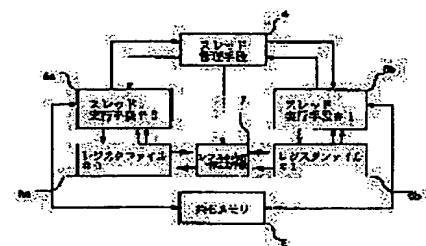
(72)Inventor : TORII ATSUSHI

(54) METHOD FOR EXECUTING MULTI-THREAD

(57)Abstract:

PROBLEM TO BE SOLVED: To reduce the overhead of a parallel processing, and to attain a parallel processing with high performance by allowing a newly generated thread to efficiently succeed the content or a register at the time of operating the parallel processing of a thread level.

SOLUTION: A sled executing means 5a executes a fork instruction, and makes a thread generating request to a thread managing means 4. The thread managing means 4 retrieves a sled executing means 5b in an execution waiting state, and makes a thread activating request to this. At the same time, the sled managing means 4 makes the copy request of a register content from a register file 6a to a register file 6b to a register content batch transferring means 7. The register content batch transferring means 7 operates the copy of the register content from the register file 6a to the register file 6b according to this request.



LEGAL STATUS

[Date of request for examination] 30.08.1996

[Date of sending the examiner's decision of rejection] 02.03.1999

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number] 2970553

[Date of registration] 27.08.1999

[Number of appeal against examiner's decision of rejection] 11-05254

[Date of requesting appeal against examiner's decision of rejection] 01.04.1999

[Date of extinction of right]

Copyright (C); 1998,2000 Japanese Patent Office

Best Available Copy

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平10-78880

(43) 公開日 平成10年(1998) 3月24日

| (51) Int.Cl. ⁸ | 識別記号 | 庁内整理番号 | F I | 技術表示箇所 |
|---------------------------|-------|--------|--------------|---------|
| G 0 6 F 9/46 | 3 6 0 | | G 0 6 F 9/46 | 3 6 0 B |
| 9/38 | 3 7 0 | | 9/38 | 3 7 0 A |
| 15/16 | 4 3 0 | | 15/16 | 4 3 0 B |

審査請求 有 請求項の数11 F D (全 32 頁)

(21) 出願番号 特願平8-249272

(22) 出願日 平成8年(1996) 8月30日

(71) 出願人 000004237

日本電気株式会社

東京都港区芝五丁目7番1号

(72) 発明者 鳥居 淳

東京都港区芝五丁目7番1号 日本電気株式会社内

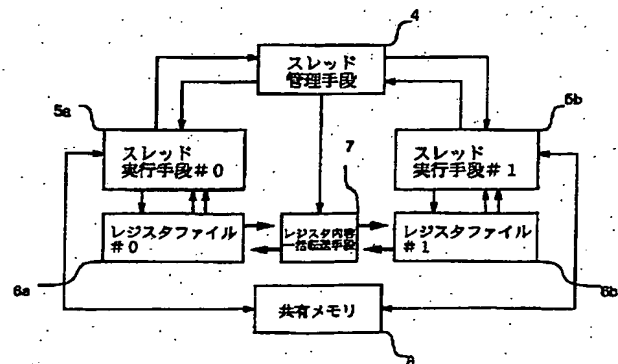
(74) 代理人 弁理士 境 廣巳

(54) 【発明の名称】 マルチスレッド実行方法

(57) 【要約】

【課題】 スレッドレベルの並列処理を行う際に、新規生成スレッドに効率的にレジスタの内容を継承させることによって、並列処理のオーバーヘッドを削減し、高性能な並列処理を可能にする。

【解決手段】 スレッド実行手段5 aはフォーク命令を実行すると、スレッド管理手段4に対してスレッド生成要求を行う。スレッド管理手段4は実行待ち状態のスレッド実行手段5 bを検索し、これに対してスレッド起動要求を行う。同時に、スレッド管理手段4は、レジスタ内容一括転送手段7に対してレジスタファイル6 aからレジスタファイル6 bへのレジスタ内容のコピー要求を行う。レジスタ内容一括転送手段7は、この要求に従いレジスタファイル6 aからレジスタファイル6 bへのレジスタ内容のコピーを行う。



【特許請求の範囲】

【請求項1】 論理的に複数のプログラムカウンタを有し、これらのプログラムカウンタに従った複数のスレッドの命令を同時にフェッチ、解釈、実行するスレッド実行手段と、独立した論理的なレジスタファイルとを備え、単一のスレッド中の指定された命令によって、新たなスレッド（子スレッド）の実行を開始する機能を備えたプロセッサにおいて、

子スレッド実行開始命令を実行するスレッド（親スレッド）の命令実行時点のレジスタファイルの内容を子スレッドのレジスタファイルに継承することを特徴としたマルチスレッド実行方法。

【請求項2】 スレッド毎に複数のレジスタファイル及びレジスタ選択手段を備えると共に、レジスタ内容転送手段を備え、

子スレッドは、前記レジスタ選択手段によって親スレッドのレジスタファイルを参照し、

レジスタの変更が行われる毎に、前記レジスタ内容転送手段によって順次新スレッドのレジスタファイルに、変更する前のレジスタ内容を転送し、

前記レジスタ選択手段の選択内容を子スレッド側のレジスタファイルに切替えることによってレジスタファイルを継承させることを特徴とした請求項1記載のマルチスレッド実行方法。

【請求項3】 親スレッドのプログラムにおける子スレッド実行開始命令とその前後に配置される他の演算命令との間で、プログラムの意味を変えない範囲内において非プログラム順序に実行することを特徴とした請求項1記載のマルチスレッド実行方法。

【請求項4】 演算の結果とレジスタ番号を指定するタグを一時的に格納するリオーダーバッファを持ち、命令をデコードした際に、そのリオーダーバッファのエントリを確保し、命令をプログラムで指定された順序ではなく、必要なレジスタの値が使用可能になったものからレジスタ及びリオーダーバッファの当該命令よりも前のエントリから供給することによって演算を行い、演算結果を命令デコード時に確保したリオーダーバッファのエントリに格納し、そのリオーダーバッファからはプログラム順序でレジスタ内容の更新を行うことによって、プログラムの順序に従わずに処理を進める非プログラム順序実行型プロセッサにおいて、

複数のプログラムカウンタを有し、これらのプログラムカウンタに従った複数のスレッドの命令を同時にフェッチ、解釈、実行するスレッド実行手段と、複数のレジスタファイルおよびリオーダーバッファを設け、

親スレッドと子スレッドの両者のレジスタファイルおよびリオーダーバッファからの内容出力を選択するレジスタデータセクタ装置を設け、

子スレッド生成命令が親スレッド内でプログラム順序で完了した時点で、親スレッドを実行しているレジスタフ

イルの内容を子スレッドを実行しているレジスタファイルにコピーすることによって、

コピー前は親スレッドのレジスタファイル、リオーダーバッファおよび子スレッドのリオーダーバッファからレジスタ内容を選択し、コピー後は子スレッドのリオーダーバッファおよびレジスタファイルからレジスタ内容を選択することによってレジスタの継承を行うことを特徴とした請求項3記載のマルチスレッド実行方法。

【請求項5】 演算の結果とレジスタ番号を指定するタグを一時的に格納するリオーダーバッファを持ち、命令をデコードした際に、そのリオーダーバッファのエントリを確保し、命令をプログラムで指定された順序ではなく、必要なレジスタの値が使用可能になったものからレジスタ及びリオーダーバッファの当該命令よりも前のエントリから供給することによって演算を行い、演算結果を命令デコード時に確保したリオーダーバッファのエントリに格納し、そのリオーダーバッファからはプログラム順序でレジスタ内容の更新を行うことによって、プログラムの順序に従わずに処理を進める非プログラム順序実行型プロセッサにおいて、

複数のプログラムカウンタを有し、これらのプログラムカウンタに従った複数のスレッドの命令を同時にフェッチ、解釈、実行するスレッド実行手段と、複数のレジスタファイルおよびリオーダーバッファを設け、

親スレッドと子スレッドの両者のリオーダーバッファ及び子スレッドのレジスタファイルからの内容出力を選択するレジスタデータセクタ装置を設け、

レジスタファイルの内容のコピーを子スレッド生成が行われた時点で行うこととして、その後は親スレッドのリオーダーバッファから、親スレッドのレジスタファイルと子スレッドのレジスタファイルに書き込みを行い、

スレッド生成命令がプログラム順序で終了する前は親スレッドのリオーダーバッファ、子スレッドのレジスタファイルおよびリオーダーバッファからレジスタ内容を選択し、プログラム順序で終了した後は子スレッドのリオーダーバッファおよびレジスタファイルからレジスタ内容を選択することによってレジスタの継承を行うことを特徴とした請求項3記載のマルチスレッド実行方法。

【請求項6】 レジスタファイルのコピーを複数回の転送サイクルによって行うレジスタ内容転送手段を用い、レジスタ内容の転送が済んだレジスタファイル部分から、新スレッドにおいて、参照を許可することを特徴とした請求項4または5記載のマルチスレッド実行方法。

【請求項7】 スレッド生成命令を実行した時点で、そのスレッドを実行できる資源が確保できない場合に、レジスタの内容を退避用レジスタファイルに蓄えることによって、プロセッサの許容数以上のスレッドが同時に存在し得るようにしたことを特徴とする請求項4、5または6記載のマルチスレッド実行方法。

【請求項8】 プログラムから指定する論理レジスタと

ハードウェアに実行される物理レジスタとの対応関係を可変とし、この対応関係を記録、更新、検索するレジスタ写像テーブルを備え、論理レジスタに対して値の書き込みを行おうとする毎に物理レジスタとの新しい対応関係を生成し、対応する物理レジスタに値の書き込みが完了した時点で読み出しを許可し、命令がプログラム順序に完了した時点で、論理レジスタとの古い対応関係になっていた物理レジスタを未使用状態にすることによって、非プログラム順序で命令を実行するプロセッサにおいて、

複数のプログラムカウンタを有し、これらのプログラムカウンタに従った複数のスレッドの命令を同時にファイル、解釈、実行するスレッド実行手段を設け、前記複数のスレッド実行手段から参照できる共有の物理レジスタファイルと、この共有の物理レジスタファイルの使用状態保持手段と、複数のレジスタ写像テーブルとを設け、

子スレッド実行開始命令を解釈した時点で、これらのレジスタ写像テーブル間で内容をコピーすることによってレジスタの継承を行い、

継承した物理レジスタを親スレッド、子スレッド両者で対応する論理レジスタに書き込みを行った時点で、未使用状態にすることによってレジスタの継承を行うことを特徴とした請求項3記載のマルチスレッド実行方法。

【請求項9】 レジスタ写像テーブルの内容のコピーを複数回の転送サイクルによって行い、

新スレッドにおいて、対応関係の転送が済んだものから使用可能にすることを特徴とした請求項8記載のマルチスレッド実行方法。

【請求項10】 スレッド生成命令を実行した時点で、そのスレッドを実行できる資源が確保できない場合に、レジスタ写像テーブルの内容を退避用レジスタ写像テーブルに蓄えることにより、

プロセッサの許容数以上のスレッドが同時に存在し得るようにすることを特徴とした請求項8または9記載のマルチスレッド実行方法。

【請求項11】 親スレッドから子スレッドを生成する回数を高々1回に制限し、スレッドの消滅順序を親スレッドを子スレッドより先にすることによって、

隣接するスレッド実行手段に限定してレジスタ内容継承を行うようにしたことを特徴とする請求項2、3、4、5、6、7、8、9または10記載のマルチスレッド実行方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は複数の命令を同時に実行する高性能なマイクロプロセッサに関し、特にマルチスレッド実行方法に関する。

【0002】

【従来の技術】プロセッサの高速化の技術として、複数

の演算装置を用意し、命令単位の並列性を利用して複数の演算装置に同時に命令を発行することにより処理速度を向上する方式が実用化されている。この方式を用いた場合、理想的には1クロックで複数の命令を実行することが可能である。

【0003】しかしながら、現実には命令間に依存関係が存在し、前の命令が終了しないと後ろの命令が実行できない場合が存在するため、同時に実行できる命令数は制限されてしまう。また、条件分岐命令によって、命令がスムーズに演算装置に供給できないという現象も生じる。これらのことから、演算装置を無限に増やした場合にも、実際には3〜4倍の性能向上に抑えられてしまうといわれている。

【0004】このような性能向上の限界は、Monica S. Lam氏が1992年に発表した論文(Monica S. Lam and Robert P. Wilsn, "Limits of Control Flow on Parallelism", IThe 19th International Symposium on Computer Architecture, IEEE Computer Society Press, 1992, pp. 46-57)に述べられている。

【0005】この限界を踏まえ、さらなる性能向上を図る手段として、

○命令単位の並列処理をさらに進めるために、非プログラム順序実行機構(out-of-order実行機構)やレジスタリネーミング機構を導入し、命令間の依存関係を少なくする。

○プログラムを複数の命令流(スレッド)に分割して、そのスレッドレベルで並列処理を行う。などの技術が提案されている。

【0006】最初の項目の、out-of-order実行機構はプログラムの実行順序に従わずに実行可能になった命令から先に実行するための機構である。このためには、レジスタ間でデータの直接の依存関係はないが、レジスタ割り付けの際にレジスタ数不足によって生じるような偽りの依存関係を解消する必要がある。このような依存に対して、ソフトウェアで指定されたレジスタの名前に対し、ハードウェアによって別のレジスタの名前に写し変えることを行うのが、レジスタリネーミング機構である。

【0007】例えば、以下のプログラム順序で命令が与えられているとする。

10 add r1←r2+r3

14 sub r4←r1-r5

18 add r5←r6+r7

1c sub r8←r4-r5

ここで、0x14番地のsub命令は、0x10番地のadd命令の結果(r1レジスタ)を用いているので、

直接のデータ依存関係があることになる。同様に 0×1 c 番地の `sub` 命令は、 $r5$ レジスタを介して 0×18 番地の `add` 命令と依存関係がある。しかしながら、 0×14 番地の `sub` 命令で読み出し参照され、 0×18 番地の `add` 命令で書き込み参照される $r5$ レジスタには、データの依存関係は存在しないが、レジスタの再使用による偽りの依存関係が存在する。従って、`out-of-order` 実行機構を備えるだけでは、 0×18 番地の `add` 命令は 0×14 番地の `sub` 命令を追い越すことはできない。レジスタリネーミング機構は、このような場合に、 0×14 番地の $r5$ と 0×18 番地の $r5$ を異なったレジスタにリネーミングすることによって、`out-of-order` 実行を可能とするものである。

【0008】このレジスタリネーミング機構は、リオーダーバッファ方式とレジスタ写像テーブル方式の2種類に大別される。図30にリオーダーバッファ方式の構成図を、図31にレジスタ写像テーブル方式の構成図を示す。

【0009】リオーダーバッファ方式は、`out-of-order` で計算された演算結果をすべてリオーダー・バッファ123と呼ばれる命令毎にエントリが確保されるバッファに一旦格納し、このリオーダー・バッファ123からプログラム順序でレジスタファイル122に書き戻しを行う。後続命令はリオーダー・バッファ123からの値をレジスタファイル122からの値に優先して用いることによって、`out-of-order` 実行のレジスタリネーミングを実現する。

【0010】また、レジスタ写像テーブル方式は、プログラムから指定する論理レジスタ番号を物理レジスタ番号にレジスタ写像テーブル124によって変換して、レジスタリネーミングを行う。レジスタに書き込みが生じる度に、レジスタフリーテーブル128から指定された未使用の物理レジスタを論理レジスタに割り付ける。また、有効命令順序バッファ126によって命令がプログラム順序で終了する度に、その命令が新たに生成する前の古い写像関係にある物理レジスタを未使用状態にする。このように物理レジスタを使い回すことによって、レジスタリネーミングを実現する。

【0011】このような機構を用いた例として、リオーダーバッファ方式は米Intel社のPentium- μ プロセッサ、レジスタ写像テーブル方式は米MIPS Technology, INC. 社のP10000プロセッサなどの例があげられる。

【0012】しかしながら、このような機構を用いた場合にも`out-of-order` 実行可能な命令の範囲は最も古い命令から16~32命令程度であり、実行可能な命令があまり存在しないことも多い。また、この範囲を現実的な範囲で増やした場合も実行できる命令よりも依存などによって実行できない命令が多く入ることに

なり、ハードウェア量増加分の性能向上は見込めない。

【0013】一方、二つ目の項目に示したスレッドレベルの並列処理方式は、命令単位の並列性ではなく、複数のスレッドの命令を並列に実行することにより演算装置の利用効率をあげて処理速度向上を図る方法である。この方法では、スレッド間では依存関係が少ないため前記の命令レベルの並列処理より性能向上が図りやすい。

【0014】このスレッドレベルの並列処理は、

○全くスレッド間に依存関係のないもの。

10 ○1スレッドの実行命令が多く、スレッド間の依存関係が少なく、ソフトウェアによって依存を解消しても性能上問題の少ないもの。

○1スレッドの実行命令が少なく、かつスレッド間の依存関係が多いため、ハードウェアによってスレッドレベル並列処理の実行支援が必要なもの。

に大別される。

【0015】スレッド間で依存が全くない場合や、依存が少なくスレッド粒度の大きいものでは、ハードウェアによるサポートはほとんど必要ない。このような場合のプロセッサの並列方法の実施例としては、平田氏らが1993年に発表した論文(平田 博章, 木村 浩三, 永峰 聡, 西澤 貞次, 鷺島 敬之, 「多重スレッド・多重命令発行を用いる要素プロセッサ・アーキテクチャ」, 情報処理学会論文誌, 1993年 Vol.34 No. 4 pp. 595-605)で提案された方法などがあげられる。この方法の実施例を図32に示す。

【0016】図32の例は、命令取得装置129, 命令解読装置130, 機能実行装置131, 命令間の依存解析装置132, 機能実行装置131をスケジューリングする命令調停装置133から構成される。命令解読装置130は命令調停装置133が命令を受け入れられる状態にあり、また、命令依存解析装置132から、命令発行可能である旨の指示を受けている場合に、命令を発行する。各々の命令解読装置130から発行された命令は、命令調停装置133によって、必要な機能実行装置131に割り当てられ、実際の実行が行われる。この命令調停装置133によって、機能実行装置131は各命令間で共有され、利用効率を向上させることが可能となる。また、命令調停装置133を機能実行装置131毎に分散することにより、命令調停装置133の単純化が可能である。

【0017】しかしながら、この方法では、スレッドの生成や演算データのスレッド間の伝達に対しては考慮されていない。従って、単一タスクを複数スレッドに分割し、そのタスクを高速化する場合に対処できるものではなかった。

【0018】単一タスクの処理を高速化するには、効率的なスレッド生成とスレッド間のデータの引き渡しが必要である。このような、細粒度スレッドの並列処理プロセッサの例として、Gurinder S. Soh

i氏らが1995年に発表した論文(Gurinder S. Sohi, Scott E. Breach and T. N. Vijaykumar, "Multiscalar Processor", The 22nd International Symposium on Computer Architecture, IEEE Computer Society Press, 1995, pp. 414-425)があげられる。

【0019】Multiscalar Processorでは、単一のプログラムをいくつかの基本ブロックの集合である「タスク」に分割し、これを並列に実行処理できるプロセッサで処理する。タスク間でのレジスタ内容の引渡しは、タスク生成コンパイラによって生成されたtask descriptorによって指定する。task descriptorでは、生成される可能性のあるレジスタを明示的に指定する。この指定をcreate maskと呼ぶ。また、最後にcreate maskに指定したレジスタを更新する命令にはフォワードビットを付加する。このように、Multiscalar Processorはコンパイラ解析能力に依存したコードによって並列実行を行う。

【0020】図33は、このMultiscalar Processorの実施例である。Multiscalar Processorはシーケンサ134、プロセッシングユニット135、結合ネットワーク136、データバンク137から構成される。プロセッシングユニット135は命令キャッシュ138、実行ユニット139、レジスタファイル140から構成され、システムに複数存在する。また、対応してデータバンク137も複数存在し、ARB (Address Resolution Buffer) 141、データキャッシュ142から構成される。複数のタスクの同時実行の管理はシーケンサ134によって行われ、各プロセッシングユニット135にタスクを割り付ける。レジスタの内容はtask descriptorの指定によって、データ生成時点でフォワードされる。

【0021】しかしながら、従来コードをスレッドレベル並列処理に変換する場合や、依存解析の難しいコードに対してはMultiscalar Processorでは性能向上は図れない。また、コードサイズがtask descriptorによって増加するという問題点も生じる。また、out-of-order実行に対応した技術でないため、既存の命令レベル並列処理による性能向上が行えず、従来技術に比しての性能向上が限られていた。

【0022】

【発明が解決しようとする課題】従来のスレッドレベル並列処理のスレッド生成技術では、明示的にレジスタの内容を継承させるか、メモリを経由して継承させるため、フォーク時に自動的にレジスタ内容は引き継がれ

なかった。この為、レジスタ依存を記述するか、メモリに対するストア/ロードを用いて新規生成スレッドに対してデータを引き継ぐ必要があった。従って、スレッドを生成する時には生成側にも被生成側にもスレッド生成にともなうデータ継承のため命令を挿入する必要があった。

【0023】また、out-of-order実行型のプロセッサでは、同期命令などについてはその正当性を保つためにin-order実行を行っているが、この場合の性能低下は顕著であった。従って、フォーク命令によってスレッドを生成し、細粒度スレッドの場合でも処理速度向上を目指す場合には、フォーク命令前と後の命令間でもout-of-order実行する必要があった。

【0024】本発明の目的はこのようなスレッドレベルの並列処理を行う際に、新規生成スレッドに効率的にレジスタの内容を継承させることによって、プログラムの並列性、演算器の使用効率を向上させ、細粒度のスレッドに対しても高性能を並列処理を可能としたマルチスレッド実行方法を提供することにある。

【0025】

【課題を解決するための手段】本発明の第1の発明は、論理的に複数のプログラムカウンタを有し、これらのプログラムカウンタに従った複数のスレッドの命令を同時にフェッチ、解釈、実行するスレッド実行手段(図2の5a, 5b)と、独立した論理的なレジスタファイル

(図2の6a, 6b)とを備え、単一のスレッド中の指定された命令によって、新たなスレッド(子スレッド)の実行を開始する機能を備えたプロセッサにおいて、子スレッド実行開始命令を実行するスレッド(親スレッド)の命令実行時点のレジスタファイルの内容を子スレッドのレジスタファイルに継承することを特徴とする(図1)。

【0026】また第2の発明は、第1の発明において、スレッド毎に複数のレジスタファイル(図4の13a, 13b)及びレジスタ選択手段(図4の12a, 12b)を備えると共に、レジスタ内容転送手段(図4の150)を備え、子スレッドは、前記レジスタ選択手段によって親スレッドのレジスタファイルを参照し、レジスタの変更が行われる毎に、前記レジスタ内容転送手段によって順次新スレッドのレジスタファイルに、変更する前のレジスタ内容を転送し、前記レジスタ選択手段の選択内容を子スレッド側のレジスタファイルに切替えることによってレジスタファイルを継承させることを特徴とする。

【0027】また第3の発明は、第1の発明において、親スレッドのプログラムにおける子スレッド実行開始命令とその前後に配置される他の演算命令との間で、プログラムの意味を変えない範囲内において非プログラム順序に実行することを特徴とする(図8)。

【0028】また第4の発明は、第3の発明において、演算の結果とレジスタ番号を指定するタグを一時的に格納するリオーダバッファを持ち、命令をデコードした際に、そのリオーダバッファのエントリを確保し、命令をプログラムで指定された順序ではなく、必要なレジスタの値が使用可能になったものからレジスタ及びリオーダバッファの当該命令よりも前のエントリから供給することによって演算を行い、演算結果を命令デコード時に確保したリオーダバッファのエントリに格納し、そのリオーダバッファからはプログラム順序でレジスタ内容の更新を行うことによって、プログラムの順序に従わずに処理を進める非プログラム順序実行型プロセッサにおいて、複数のプログラムカウンタを有し、これらのプログラムカウンタに従った複数のスレッドの命令を同時にフェッチ、解釈、実行するスレッド実行手段（図9の21a, 21b）と、複数のレジスタファイル（図9の25a, 25b）およびリオーダバッファ（図9の24a, 24b）を設け、親スレッドと子スレッドの両者のレジスタファイルおよびリオーダバッファからの内容出力を選択するレジスタデータセクタ装置（26a, 26b）を設け、子スレッド生成命令が親スレッド内でプログラム順序で完了した時点で、親スレッドを実行しているレジスタファイルの内容を子スレッドを実行しているレジスタファイルにコピーすることによって、コピー前は親スレッドのレジスタファイル、リオーダバッファおよび子スレッドのリオーダバッファからレジスタ内容を選択し、コピー後は子スレッドのリオーダバッファおよびレジスタファイルからレジスタ内容を選択することによってレジスタの継承を行うことを特徴とする。

【0029】また第5の発明は、第3の発明において、演算の結果とレジスタ番号を指定するタグを一時的に格納するリオーダバッファを持ち、命令をデコードした際に、そのリオーダバッファのエントリを確保し、命令をプログラムで指定された順序ではなく、必要なレジスタの値が使用可能になったものからレジスタ及びリオーダバッファの当該命令よりも前のエントリから供給することによって演算を行い、演算結果を命令デコード時に確保したリオーダバッファのエントリに格納し、そのリオーダバッファからはプログラム順序でレジスタ内容の更新を行うことによって、プログラムの順序に従わずに処理を進める非プログラム順序実行型プロセッサにおいて、複数のプログラムカウンタを有し、これらのプログラムカウンタに従った複数のスレッドの命令を同時にフェッチ、解釈、実行するスレッド実行手段（図15の50a, 50b）と、複数のレジスタファイル（図15の54a, 54b）およびリオーダバッファ（53a, 53b）を設け、親スレッドと子スレッドの両者のリオーダバッファ及び子スレッドのレジスタファイルからの内容出力を選択するレジスタデータセクタ装置（55a, 55b）を設け、レジスタファイルの内容のコピー

を子スレッド生成が行われた時点で行うこととして、その後は親スレッドのリオーダバッファから、親スレッドのレジスタファイルと子スレッドのレジスタファイルに書き込みを行い、スレッド生成命令がプログラム順序で終了する前は親スレッドのリオーダバッファ、子スレッドのレジスタファイルおよびリオーダバッファからレジスタ内容を選択し、プログラム順序で終了した後は子スレッドのリオーダバッファおよびレジスタファイルからレジスタ内容を選択することによってレジスタの継承を行うことを特徴とする。

【0030】また第6の発明は、第4または第5の発明において、レジスタファイルのコピーを複数回の転送サイクルによって行うレジスタ内容転送手段（図17の60, 61等）を用い、レジスタ内容の転送が済んだレジスタファイル部分から、新スレッドにおいて、参照を許可することを特徴とする。

【0031】また第7の発明は、第4, 第5または第6の発明において、スレッド生成命令を実行した時点で、そのスレッドを実行できる資源が確保できない場合に、レジスタの内容を退避用レジスタファイル（図19の72）に蓄えることによって、プロセッサの許容数以上のスレッドが同時に存在し得るようにしたことを特徴とする。

【0032】また第8の発明は、第3の発明において、プログラムから指定する論理レジスタとハードウェアに実行される物理レジスタとの対応関係を可変とし、この対応関係を記録、更新、検索するレジスタ写像テーブルを備え、論理レジスタに対して値の書き込みを行おうとする毎に物理レジスタとの新しい対応関係を生成し、対応する物理レジスタに値の書き込みが完了した時点で読み出しを許可し、命令がプログラム順序に完了した時点で、論理レジスタとの古い対応関係になっていた物理レジスタを未使用状態にすることによって、非プログラム順序で命令を実行するプロセッサにおいて、複数のプログラムカウンタを有し、これらのプログラムカウンタに従った複数のスレッドの命令を同時にフェッチ、解釈、実行するスレッド実行手段（図20の73a, 73b）を設け、前記複数のスレッド実行手段から参照できる共有の物理レジスタファイル（図20の78）と、この共有の物理レジスタファイルの使用状態保持手段（図20の81, 82）と、複数のレジスタ写像テーブル（図20の76a, 76b）とを設け、子スレッド実行開始命令を解釈した時点で、これらのレジスタ写像テーブル間で内容をコピーすることによってレジスタの継承を行い、継承した物理レジスタを親スレッド、子スレッド両者で対応する論理レジスタに書き込みを行った時点で、未使用状態にすることによってレジスタの継承を行うことを特徴とする。

【0033】また第9の発明は、第8の発明において、レジスタ写像テーブルの内容のコピーを複数回の転送サ

イクルによって行い、新スレッドにおいて、対応関係の転送が済んだものから使用可能にすることを特徴とする。

【0034】また第10の発明は、第8または第9の発明において、スレッド生成命令を実行した時点で、そのスレッドを実行できる資源が確保できない場合に、レジスタ写像テーブルの内容を退避用レジスタ写像テーブル（図24の98）に蓄えることにより、プロセッサの許容数以上のスレッドが同時に存在し得るようにすることを特徴とする。

【0035】また第11の発明は、第2ないし第10の発明において、親スレッドから子スレッドを生成する回数を高々1回に制限し、スレッドの消滅順序を親スレッドを子スレッドより先にすることによって、隣接するスレッド実行手段に限定してレジスタ内容継承を行うようにしたことを特徴とする。

【0036】第1の発明ではフォーク命令を行った時点で、新規生成スレッドに対してレジスタ内容が継承される。この方法を用いる場合には、レジスタに格納できる範囲内のデータ数であれば、メモリに対するストア/ロードを省くことが可能になる。

【0037】また第2の発明では、複数のレジスタファイルを選択する手段を用いて、レジスタファイルの内容の物理的なコピーを行わずに、レジスタ内容の継承を実現できる。

【0038】また第3の発明では、プログラム順序において新規スレッドを生成する命令より前の命令の実行完了を待たずに、新規スレッドを生成する。また、新規スレッド生成命令より後続の命令がスレッド生成を行う前に実行完了できる。そして、このときに継承させるレジスタの内容はプログラム順序における新規スレッド実行命令時の内容とする処理方法を採用する。このような処理方法を実現するため、第4から第7の発明では、リオーダーバッファを用いたout-of-order型実行を行い、レジスタファイルの内容をコピーするタイミングを工夫し、コピーが終了する前や、値が確定するまでは異なったレジスタファイルやリオーダーバッファをアクセスするようにして、レジスタ内容を継承する。フォーク命令がプログラム順序に従ってレジスタ内容を更新した際に、レジスタの内容を確定させその後は新スレッド用のレジスタを用いることによってレジスタ内容を継承する。また、第8から第10までの発明では、ソフトウェアによって指定する論理レジスタを物理レジスタに写像するためのレジスタ写像テーブルを用いたレジスタリネーミングを行い、このレジスタ写像テーブルを物理的に同時に実行するスレッド数に応じて複数用意する。新たなスレッドが生成される際には、この写像情報をコピーする。プロセッサに存在する物理的なレジスタはすべてスレッドから共有され、この写像情報によって各々のスレッドが使用する。

【0039】以下に図面を参照して本発明をより具体的に説明するが、以下の開示は本発明の一実施例に過ぎず、本発明の技術的範囲をなんら限定するものではない。

【0040】

【発明の実施の形態】図1は本発明の第1の発明におけるレジスタ内容の継承方法の概念図である。図1に示すように、新スレッドを生成するスレッド（親スレッド）1がその実行フローの実行途中においてスレッド生成命令（フォーク命令）2を実行して新スレッド（子スレッド）3を生成すると、子スレッド3のレジスタファイルに親スレッド1がフォーク命令2を実行した時点の内容を継承させる。

【0041】図2は上記のようなレジスタ内容の継承方法を実現する2スレッド並列実行型プロセッサの実施例のブロック図である。この例のプロセッサは、スレッド管理手段4、スレッド実行手段5a、5b、レジスタファイル6a、6b、レジスタ内容一括転送手段7および共有メモリ8から構成される。

【0042】スレッド管理手段4は、プロセッサ全体のスレッドの実行管理を行う。また、スレッド実行手段5a、5b、レジスタファイル6a、6bは同時実行可能なスレッド数と同数用意され、通常のプロセッサ同様のプログラムコード中の命令を実行し、レジスタファイル6a、6bおよび共有メモリ8に結果を書き込みながら処理を進める。

【0043】以下、スレッド生成時の動作を時間順序を追って説明する。図3は図2に示した実施例のプロセッサにおけるスレッド生成時のタイミングチャートである。図3において、(A)はプロセッサ全体のクロック、(B)はスレッド実行手段#0 5aの実行状態、(C)はレジスタファイル#0 6aのアクセス状態、(D)はスレッド実行手段#1 5bの実行状況、(E)はレジスタファイル#1 6bのアクセス状態、(F)はスレッド管理手段4のアクセス状況を示す。

【0044】図3のタイミングチャートの開始時点ではスレッド実行手段#0 5aにおいてのみ処理が行われており、スレッド実行手段#1 5bは実行待ち状態である。従ってレジスタファイル#1 6bに対するアクセスは生じない。

【0045】サイクル6において、スレッド実行手段#0 5aはフォーク命令を実行すると、スレッド管理手段4に対してスレッド生成要求を行う。スレッド管理手段4は実行待ち状態のスレッド実行手段を検索し、新しいスレッドを実行するスレッド実行手段を決定する。図3の場合、(D)で示されるスレッド実行手段#1 5bが実行待ち状態であるので、ここで新しいスレッドを実行することを決定し、スレッド管理手段4はスレッド実行手段#1 5bに対してスレッド起動要求を行う。同時に、スレッド実行手段4は、レジスタ内容一括転送手

段7に対してレジスタファイル#0 6aからレジスタファイル#1 6bへのレジスタ内容のコピー要求を行う。レジスタ内容一括転送手段7は、この要求に従いレジスタファイル#0 6aからレジスタファイル#1 6bへのレジスタ内容のコピーを行う。これらの動作によって、サイクル7からは、二つのスレッドが並列に実行され、子スレッド3では、初期値を共有メモリ8からロードすることによってレジスタファイル6にセットする必要がなくなり、親スレッド1では必要なデータをレジスタファイル6から共有メモリ8にストアする必要がなくなり、効率的な新スレッド生成が可能になる。なお、並列動作ではレジスタファイル6a、6bは独立に参照、更新されることになる。

【0046】次に、第2の発明について説明する。図4はその実施例の2スレッド並列実行型プロセッサのブロック図であり、スレッド実行手段を9a、9bの二つ持ち、各々のスレッド実行手段9a、9bは、命令キャッシュ10a、10b、命令デコーダ11a、11b、レジスタ選択手段12a、12b、レジスタファイル13a、13b、演算ユニット14a、14bを各々持っている。また、共有のスレッド管理手段15とレジスタ内容転送手段150が存在する。なお、共有メモリについては直接関係しないので、図4からは省いてある。

【0047】図5はレジスタ選択手段#1 12bの構成例を示すブロック図である。レジスタ選択手段#0 12aも同等の構造であり、レジスタ選択テーブル16に各レジスタの選択ビット17を持っている。本実施例の場合、レジスタ選択ビット17が0の場合にはレジスタファイル#0 13aを、1の場合にはレジスタファイル#1 13bを選択する。

【0048】図6はレジスタファイル#0 13aの構成例を示すブロック図であり、レジスタファイル#1 13bも同等の構造である。レジスタファイル#0 13aは、各レジスタ毎にレジスタ内容メモリ18の他に、転送終了ビット19、更新ビット20を持つ。転送終了ビット19は、スレッド管理手段15によって、子スレッド生成時に初期化(リセット)され、レジスタ内容メモリ18の内容を他のレジスタファイル13に転送した際にセットされる。また、更新ビット20は、スレッド起動時に初期化(リセット)され、レジスタを自分の属する演算ユニット14で更新した場合にセットされる。

【0049】以下、スレッド生成時のレジスタ内容の継承方法を具体例を用いて説明する。図7は、図4に示した実施例のプロセッサのタイミングチャートである。図7において、(A)はプロセッサ全体のクロック、

(B)は演算ユニット#0 14a(親スレッド)で実行している命令もしくは書き込みされるレジスタ番号、(C)は親スレッドによるレジスタファイル#0 13aに対するレジスタ内容の書き込み、(D)はレジスタ

内容の転送によるレジスタファイル#0 13aに対するレジスタ内容の読み出し、(E)はレジスタファイル#0 13a中の転送終了ビット19の状態、(F)はスレッド管理手段15のアクセス状況、(G)は演算ユニット#1 14b(子スレッド)で実行している命令もしくは読み出し/書き込みされるレジスタ番号、

(H)、(I)は子スレッドによるレジスタファイル13の読み出し、(J)はレジスタファイル#1 13bへの書き込み、(K)はレジスタファイル#1 13b中の更新ビット20の状態、(L)はレジスタ選択手段#1 12b中のレジスタ選択ビット17の状態を示す。

【0050】図7のタイミングチャートの開始時点では、スレッドはスレッド実行手段#09aのみで実行されている。このスレッドを親スレッドとしている。サイクル2において、親スレッドはフォーク命令を実行する。スレッド管理手段15は、レジスタファイル#0 13aの転送終了ビット19と、レジスタファイル#1 13bの更新ビット20と、レジスタ選択手段#1 12b中のレジスタ選択ビット17の初期化要求を行う。このフォーク命令の実行によってサイクル3以降はサイクル2のフォーク命令による子スレッドがスレッド実行手段#1 9bで実行が開始される。

【0051】サイクル3において親スレッドを実行しているスレッド実行手段#0 9aではレジスタr1に対する書き込みが生じる前に、スレッド実行手段#1 9bにr1のレジスタ内容を転送するために読み出しが行われる。これがサイクル3前半の(D)に示されている。この内容はスレッド実行手段#1 9bのレジスタファイル#1 13bに転送され(J)で示されるように書き込みが行われる。同時に、レジスタファイル#0 13aの転送終了ビット19と、レジスタファイル#1 13bの更新ビット20のレジスタr1のエントリをセットし、レジスタ選択手段#1 12b中のレジスタ選択ビット17をr1またはレジスタファイル#1 13bから選択するようにセットする。

【0052】また、同サイクルにおいて子スレッドを実行しているスレッド実行手段#1 9bではレジスタr7、r10を読み出し参照する。この読み出しはレジスタ選択手段#1 12bの内容に従ってレジスタファイル#0 13aから読み出す。また、同サイクルにおける子スレッドはr3に対して書き込みを行うが、この際にレジスタ選択手段#1 12b中のレジスタ選択ビット17のr3をレジスタファイル#1 13bから選択するようにセットする。同時にレジスタファイル#1 13bの更新ビット20のr3エントリをセットする。

【0053】サイクル4も同様の処理が行われ、r5の内容が転送され、レジスタファイル#0 13aの転送終了ビット19と、レジスタファイル#1 13bの更新ビット20と、レジスタ選択手段#1 12b中のレ

レジスタ選択ビット17のレジスタr5のエントリがセットされる。また、同サイクルにおいてレジスタ選択手段#1 12b中のレジスタ選択ビット17と、レジスタファイル#1 13bの更新ビット20のr2エントリをセットする。

【0054】次に、サイクル5では親スレッドを実行しているスレッド実行手段#0 9aでは再びレジスタr1に対する書き込みを行う命令を実行する。しかしながら、レジスタファイル#0 13aの転送終了ビット19のr1エントリは既にセットされているので、レジスタ内容の転送は行われぬ。また、同サイクルにおいて子スレッドを実行しているスレッド実行手段#1 9bではレジスタr3, r5を読み出し参照する。この読み出しはレジスタ選択手段#1 12bの該当エントリはセットされているのでレジスタファイル#1 13bから読み出す。

【0055】次に、サイクル6では親スレッドを実行しているスレッド実行手段#0 9aではレジスタr2に対する書き込みを行う命令を実行し、レジスタの内容が転送される。しかしながら、レジスタファイル#1 13bの更新ビット20のr2エントリは既にセットされているので、レジスタファイル#1 13bのレジスタ内容メモリ18には書き込みは生じない。

【0056】以上の処理によって、スレッド生成のフォーク命令実行時にレジスタファイルの全内容を一括転送する必要がなくなり、レジスタファイル間の転送バンド幅を削減しつつ、レジスタ内容を子スレッドに継承することが可能となる。

【0057】本実施例は2スレッド並列実行型プロセッサについて説明したが、3スレッド以上の並列処理を行う際には、スレッド管理ユニットによってレジスタの転送先を制御し、レジスタファイル間はバスなどで接続し多対多転送をサポートするような拡張を施せば良い。また、レジスタ選択手段ではどのレジスタファイルを選択するかを示すレジスタ選択ビットが複数ビットに格納されるが、本質的にレジスタ選択手段を用いることには相違なく、本発明の範囲内である。

【0058】次に第3の発明について説明する。第3の発明は、ソフトウェアによるプログラミングモデルでは第1の発明の方法に従うが、ハードウェアによる実際の処理は図8に示すように必ずしもプログラム順序に実行を行わないハードウェア処理方法である。つまり、子スレッドには図8で示されたパイプラインの実行イメージ図において、0x00番地、0x04番地の命令実行によるレジスタ内容を継承させ、0x0c番地、0x10番地の命令実行によるレジスタ内容は継承しない処理方法である。第4の発明以降はこの処理方法に基づく実施形態についての発明である。

【0059】次に第4の発明について説明する。図9はその実施例のブロック図であり、例として2スレッド並

列実行型プロセッサを採り上げている。図9のプロセッサは、スレッド実行手段を#0 21aと#1 21bの二つ持ち、各々は命令キャッシュ22a, 22b, 命令デコーダ23a, 23b, リオーダバッファ24a, 24b, レジスタファイル25a, 25b, レジスタデータセクタ装置26a, 26b, 命令キュー27a, 27b, 演算ユニット28a, 28bを持っている。また、共有のスレッド管理手段29が存在する。

【0060】図10は、命令キュー#0 27aの構成例を示すブロック図であり、命令キュー#1 27bも同様の構造である。命令キュー27は、デコードした命令について、必要なレジスタの値が確定し演算できるようになるまで待ち合わせる機構であり、キューエントリ格納論理30, 発行命令決定論理31, 命令キューエントリ32から構成される。

【0061】命令キューエントリ32は、命令発行に必要な情報を蓄えておくもので、エントリ有効ビット33, レジスタ内容/リオーダバッファタグ格納メモリ34, レジスタ内容有効ビット35, 結果書き込み場所指定タグ36, 命令コード格納メモリ37のエントリから構成される。エントリ有効ビット33はそのエントリに格納されているデータが有効かどうかを示す。レジスタ内容/リオーダバッファタグ格納メモリ34は、レジスタ内容有効ビット35の状態によって格納されている値が異なる。もし、セットされている場合には対応する命令の演算に必要なデータ、セットされていない場合にはリオーダバッファ24が付加したタグ番号である。また、結果書き込み場所指定タグ36はこの命令の実行による結果を格納するリオーダバッファ24のタグが格納され、命令コード格納メモリ37のエントリは、命令デコーダ23でデコードされた命令種類のコードが格納される。

【0062】キューエントリ格納論理30は、空いているエントリに対して必要なデータを格納する論理であり、エントリ有効ビット33の無効なエントリのうち一つを決定し、レジスタデータセクタ装置26, 命令デコーダ23からの情報を格納する。発行命令決定論理31は、レジスタ内容有効ビット35が有効になったエントリの中から、発行する命令を決定し、演算ユニット28に対してその命令を発行する。

【0063】また命令キューエントリ32は、連想メモリ形態となっており、レジスタ内容有効ビット35がセットされていないエントリに対して、演算ユニット28から送られてくるタグ番号と自分の持つタグ番号を比較して、同一の場合には、演算結果をレジスタ内容/リオーダバッファタグ格納メモリ34に格納し、レジスタ内容有効ビット35をセットする。

【0064】図11は、リオーダバッファ#0 24aの構成例を示すブロック図であり、リオーダバッファ#1 24bも同様の構造である。リオーダバッファ24

は、プログラム順序に従わずに先行して確定したレジスタ値を保持しておき、プログラム順序に従ってレジスタファイル25に書き戻す機構であり、レジスタデータ供給決定論理38、エントリシフト制御論理39、リオーダーバッファエントリ40から構成される。

【0065】リオーダーバッファエントリ40は、そのエントリに格納されたデータが有効かどうかを示すエントリ有効ビット41、演算ユニット28からの結果の書き戻しの際に用いる結果書き込み指定タグ42、レジスタファイル25への書き戻し場所を指定するための、レジスタ番号43、フォーク命令後でレジスタデータを子スレッドに継承させる必要を示すスレッド生成ビット44、演算ユニット28からの結果を格納する演算データ格納メモリ46と格納されるとセットされる演算データ格納メモリ有効ビット45のエントリから構成される。

【0066】リオーダーバッファ24は、命令デコーダ23から命令を受け取ると、プログラムの命令順序に従ってエントリが確保される。つまりエントリはプログラム命令順に整列していることになる。また、エントリが確保できない場合には、命令デコードが停止する。また、フォーク命令実行後の命令を格納する際には、スレッド生成ビット44をセットする。

【0067】また、命令デコーダ23からソースレジスタの参照番号も同時に受け取り、これとリオーダーバッファエントリ40中のレジスタ番号43を比較する。同じエントリが存在すれば、その中で最も新しいエントリの演算データ格納メモリ46か結果書き込み指定タグ42の内容をレジスタデータセクタ装置26に送る。演算データ格納メモリ46の内容が送られる場合は、該当エントリの演算が終了しており、演算データ格納メモリ有効ビット45がセットされている場合である。また、他のスレッド実行手段21のレジスタデータセクタ26に送る場合には、スレッド生成ビット44がセットされていないエントリから選択する。これによって、フォーク命令後の親スレッドのレジスタ変更データが子スレッドに送られることを防ぐ。これらの処理をレジスタデータ供給決定論理38が行う。

【0068】演算結果が演算ユニット28から来た場合には、結果書き込み指定タグ42の一致するエントリの演算データ格納メモリ46に書き込み、演算データ格納メモリ有効ビット45をセットする。また、演算が終わったものをプログラム順序に従ってレジスタファイル25に書き戻し、エントリシフト制御論理39によって、先頭からエントリをシフトする。なお、シフトを行わずにリング状のバッファ管理を行うことも可能である。また、フォーク命令がプログラム順序で完全に終了した場合には、すべてのエントリのスレッド生成ビット44をリセットする。

【0069】レジスタデータセクタ装置#1 26bは、図12に示す論理によって、命令キュー#1 27

bに供給するデータをリオーダーバッファ#1 24b、レジスタファイル#1 25b、リオーダーバッファ#0 24a、レジスタファイル#0 25aの入力データから選択する。レジスタデータ選択装置#0 26aも同様の論理である。

【0070】以下、図9のプロセッサの実際の動作について説明する。図9のプロセッサはスレッド生成時以外の通常時はスレッド実行手段21a、21bでそれぞれ独立して処理を進める。従って、レジスタデータセクタ装置26には、他方のスレッド実行手段21のリオーダーバッファ24、レジスタファイル25からデータが供給されることはなく、自らのスレッド実行手段21のリオーダーバッファ24、レジスタファイル25からのデータを選択して、レジスタデータが使用可能になった命令から実行が行われる。この点においては、リオーダーバッファを用いてout-of-order実行を行う従来のスーパースカラプロセッサの処理方法と相違はない。

【0071】フォーク時の動作モデルを図13に示す。フォーク命令がスレッド実行手段#0 21aでデコードされると、スレッド管理手段29によってスレッド実行手段#1 21bで子スレッドの実行が開始される。しかしながら、レジスタファイル#1 25bには、親レジスタのデータが継承されていないので、レジスタファイル#0 25aとリオーダーバッファ#0 24aのデータをレジスタデータセクタ装置#1 26bで選択して用いて演算を行うことになる。この演算の結果はリオーダーバッファ#1 24bに格納される。従って、後続命令は、レジスタファイル#0 25aとリオーダーバッファ#0 24aとリオーダーバッファ#1 24bのデータをレジスタデータセクタ装置#1 26bで選択して用いて演算を行う。

【0072】一方、親スレッド側でフォーク命令がプログラム順序で完全終了した時点で、レジスタファイル#0 25bのすべての値がフォーク命令実行時の値と確定する。この時に、スレッド管理手段29によってレジスタファイル#0 25aの内容をレジスタファイル#1 25bにコピーする。この動作が終了すると、レジスタファイル#1 25bには親レジスタの値が継承されたことになるので、レジスタファイル#1 25bとリオーダーバッファ#1 24bのデータをレジスタデータセクタ装置#1 26bで選択して用いて演算を行うことになる。また、この時点で、リオーダーバッファ#1 24bのレジスタファイル#1 25bへのレジスタアップデート動作が可能になる。これによって、out-of-order実行を行うプロセッサにおいてもレジスタ内容の継承が可能になる。

【0073】以上が図9に示した2スレッド並列実行型プロセッサの実施例の説明である。

【0074】本実施例の他の構成として、命令キュー27、演算ユニット28をスレッド間で共有する手法があ

げられる。図14は4スレッド並列実行2演算ユニット型プロセッサの場合の実施例のブロック図である。図14では、命令キャッシュ、命令デコーダ、スレッド管理手段等は省略しているが、この部分の実際の構成は図9と同様である。本実施例では新たに、リオーダバッファ24への結果振り分け装置47a、47bを用意し、演算ユニット28からの結果を振り分けている。このため、命令キュー49には、どの命令デコーダによってデコードされた命令であるかを記憶するエントリの追加が必要である。また、レジスタデータ選択装置48a、48bは、選択するレジスタ値がどの命令デコーダの命令について取り扱っているかの情報が必要となる。また、4スレッド構成としたので、レジスタファイル25の転送手段が多対多転送を実現するような構成にする必要があり、具体的にはバスなどを用いる必要がある。

【0075】次に第5の発明について説明する。図15は、その実施例のブロック図であり、例として2スレッド並列実行型プロセッサを採り上げている。主たる構成は、図9と同様である。構成上の相違点は、リオーダバッファ53から他のスレッド実行手段50のレジスタファイル54に対してレジスタ内容更新用のパスが存在すること、及びレジスタデータセレクト装置55への入力に他のスレッド実行手段50のレジスタファイル54からの入力が存在しないことである。

【0076】フォーク時の動作モデルを図16に示す。親スレッドでフォーク命令がスレッド実行手段#0 50aでデコードされると、スレッド管理手段58によってスレッド実行手段#1 50bで子スレッドの実行が開始され、同時にスレッド管理手段58によってレジスタファイル#0 54aの内容をレジスタファイル#1 54bにコピーする。

【0077】しかしながら、レジスタファイル#0 54aでは、フォーク命令よりも前の命令によるレジスタ更新が完全に行われておらず、更新データが親スレッドのスレッド実行手段50のリオーダバッファ#0 53aに存在するか、まだ演算が行われていない可能性がある。従って、レジスタデータセレクト装置#1 55bでは、子スレッドのレジスタファイル#1 54bとリオーダバッファ#0 53aとリオーダバッファ#1 53bのデータを選択して用いて演算を行う。

【0078】一方、親スレッド側でフォーク命令がプログラム順序で完全終了した時点で、リオーダバッファ#0 53aのデータをレジスタファイル#1 54bに供給する必要はなくなるので、レジスタファイル#1 54bとリオーダバッファ#1 53bのデータをレジスタデータセレクト装置#1 55bで選択して用いて演算を行うことになる。また、この時点で、リオーダバッファ#1 53bのレジスタファイル#1 54bへのレジスタアップデート動作が可能になりout-of-order実行を行うプロセッサにおいてもレジスタ

内容の継承が実現される。

【0079】図9、図15に示したプロセッサでは、レジスタファイルの内容のコピーをフォーク命令の完了時/デコード時に一度行うことを前提にしたレジスタファイル間の転送手段を前提にしていた。しかしながら、現実にはレジスタファイルの内容のすべてを同時にコピーするには、高バンド幅の転送手段が必要となる。

【0080】第6の発明は、このような問題に対して、レジスタファイルの転送を複数回に分割することによって対処するものである。図17は、第6の発明の実施例のブロック図である。同図は、16ワードレジスタファイル、4ワード4回転送型レジスタ転送手段の構成例を示しており、送信側のレジスタファイル#0 59a、受信側のレジスタファイル#1 59b、マルチプレクサ60、デバイダ61、参照許可ビット62a、62bから構成される。

【0081】図18は、図17の実施例のタイミングチャートである。図18において、(A)から(D)は転送中のレジスタファイルを示し、(E)はマルチプレクサ60からデバイダ61へ現在どのレジスタを転送しているかを伝達するための情報線の内容を示し、(F)から(I)はレジスタファイル#1 59bの参照許可ビット62bの状態を示している。

【0082】レジスタファイル59のコピーが開始されると、レジスタファイルはr0-r3、r4-r7、r8-r11、r12-r15の4回に分割して転送される。転送中は送信側のレジスタファイル#0 59aの更新と、レジスタファイル#1 59bの参照は禁止される。転送が終了次第順次レジスタファイル#0 59aの更新とレジスタファイル#1 59bの参照が許可される。この参照の許可は参照許可ビット62a、62bをセットすることによって行われる。

【0083】これによって、レジスタ転送のバンド幅を低減しながらレジスタ転送が可能になる。なお、レジスタ転送時にレジスタファイルのアクセスが禁止されるサイクルが増加するが、順次レジスタアクセス禁止が解除されるので、その特性に合わせたコードスケジュールを行い、使用できないレジスタに対するアクセスを遅らせることにより、このレイテンシを或る程度隠蔽することが可能になる。

【0084】次に、第7の発明について説明する。図19はその実施例のブロック図である。基本的には、第4の発明にかかる図9の実施例とほぼ同様であるが、退避用レジスタファイル72が新たに設けられている。

【0085】空いているスレッド実行手段63が存在しない時にさらにフォーク命令が実行された場合、フォーク命令を実行したスレッド実行手段63がレジスタファイル67の確定した時にこの退避用レジスタファイル72にコピーを行う。このことによって、システム管理ソフトウェアの介入なく、スレッド実行手段63の数を超

えるスレッド生成要求に対処する。スレッド実行手段63に空きができた時点で、退避用レジスタファイル72から、空いたスレッド実行手段63のレジスタファイル67に対してコピーを行い、退避していたスレッドの実行を再開する。これらの管理はスレッド管理手段71によって行われる。

【0086】次に、第8の発明について説明する。図20は、その実施例の一例を示す2スレッド並列実行型プロセッサのブロック図である。図20のプロセッサはスレッド実行手段73a、73bと共有の物理レジスタファイル78、レジスタビジューテーブル81、レジスタフリーテーブル82、スレッド管理手段83から構成される。

【0087】スレッド実行手段73a、73bはそれぞれ、命令キャッシュ74a、74b、命令デコーダ75a、75b、レジスタ写像テーブル76a、76b、命令キュー77a、77b、演算ユニット79a、79b、有効命令順序バッファ8a、80bから構成される。

【0088】この実施例では、レジスタをソフトウェアからアクセスする論理レジスタと、ハードウェア的にレジスタ内容を保持する物理レジスタを分離し、その写像関係をレジスタ写像テーブル76に保持する。図21は、レジスタ写像テーブル76aの詳細なブロック図である。レジスタ写像テーブル76bも同様の構造である。レジスタ写像テーブル76は、論理レジスタ数分の物理レジスタ番号エントリを持っており、論理レジスタ番号を物理レジスタ番号に変換する。

【0089】どのスレッド実行手段73が、どのレジスタを使用しているかという情報を管理するのが、レジスタフリーテーブル82である。図22は、レジスタフリーテーブル82の詳細なブロック図である。レジスタフリーテーブル82は、レジスタフリー決定論理84、フリーレジスタ検索論理85、状態テーブル86から構成される。

【0090】命令が命令デコーダ75によってデコードされると、読み出し参照する論理レジスタ番号と書き込み参照する論理レジスタ番号が確定する。読み出し参照を行う論理レジスタ番号は、直ちにレジスタ写像テーブル76によって物理レジスタ番号に変換され、命令キューに格納される。

【0091】書き込み参照する論理レジスタに対しては、新しい物理レジスタを確保して用いる。これは、out-of-order実行を行った際に、現在デコードした命令により前の命令が同じ論理レジスタを参照する際の正当性を維持するためである。例えば、

```
10      add      r1 ← r2 + r3
14      sub      r3 ← r4 - r2
```

という命令をout-of-orderで実行する場合には、14番地の命令を実行して、更新されたr3の値

を10番地の命令が読み出すとプログラムの正当性が維持できなくなる。そこで、10番地のr3と14番地のr3を異なった物理レジスタに写像することによって、14番地の命令を実行しても実行前のr3の値を保持しておき、10番地の実行時には前のレジスタ写像関係から前のr3の値を読み出し参照するというを行う。このため、現在未使用の物理レジスタをレジスタフリーテーブル82から確保し、レジスタ写像テーブル76の写像情報を新しい論理レジスタ-物理レジスタ対応関係に更新する。

【0092】命令デコード時には、有効命令順序バッファ80に対しても命令のプログラム順序に従って必要な情報を記憶させる。ここでは、デコード命令によって新しい論理レジスタ-物理レジスタ写像関係が生成された場合、その命令実行前のレジスタ写像関係を併せて記憶させる。これは、物理レジスタの開放や処理の取消時に必要になるためである。

【0093】新たに確保した物理レジスタは実際に書き込みが生じるまでは読み出し参照を禁止する。この書き込みが生じたか否かをレジスタビジューテーブル81で管理する。確保した物理レジスタは最初ビジュー状態で読み出し参照を禁止する。その後、書き込みが生じた後にフリー状態に変更して読み出し参照を許可する。図23はレジスタビジューテーブル81の詳細なブロック図である。レジスタビジューテーブル81の管理するレジスタ数は物理レジスタ本数に対応する。

【0094】物理レジスタはプログラム順序で命令を完全に終了させる際に開放を行う。例えば、

```
10      add      r1 ← r2 + r3
14      sub      r3 ← r4 - r2
```

の例の場合、14番地の命令が終了時に10番地までに用いていたr3に対応する物理レジスタを開放する。14番地で確保したr3の物理レジスタを開放するのは、その後のr3を更新する命令がプログラム順序で完全終了する際である。但し、この開放は子スレッドにレジスタ内容が継承されていない場合である。子スレッドに継承されている場合、親スレッド、子スレッド両者が開放された時に完全に開放されたことになる。つまり、状態テーブル86のすべてのエントリがリセットされた場合に、当該レジスタは空き状態になる。

【0095】以下、図20のプロセッサの実際の動作について説明する。図20のプロセッサは、スレッド生成と終了時以外はスレッド実行手段73で独立して処理を進める。物理レジスタファイル78、レジスタビジューテーブル81、レジスタフリーテーブル82は共有であるが、スレッド毎に異なった要求を同時に処理するだけである。従って、この点においては、レジスタ写像テーブル76を用いてレジスタリネーミングを行う従来のスーパースカラプロセッサの処理方法と何ら相違はない。

【0096】さて、図20のプロセッサでは、フォーク

命令をデコードすると、レジスタ写像テーブル76の内容を他の空いているスレッド実行手段73のレジスタ写像テーブル76にコピーする。同時にレジスタフリーテーブル82に対してもコピーを行う。この情報を受けたレジスタフリーテーブル82は、子スレッドのスレッド実行手段における状態テーブルをレジスタ使用中にセットする。レジスタ写像テーブル76はデコード時にプログラム順序に従って内容が変更されるので、フォーク命令デコード時に、正しい写像情報を保持している。従って、子スレッドからも親スレッド同様に通常の物理レジスタファイル78の参照が可能である。また、スレッド生成後は同一論理レジスタへの書き込みを行った物理レジスタマッピングするので、各々のスレッドで独立した処理が可能である。スレッド終了時には、レジスタフリーテーブル82の当該スレッド実行手段73の状態テーブル86をクリアする。

【0097】このように本実施例では写像関係をコピーすることによって、out-of-orderにおけるレジスタ内容の継承を実現する。本実施例では、第4から第7までの発明に比して、コピーする情報量が少ない。また、親スレッド側でフォーク命令がプログラム順序で終了するまでの特別な制御が不要であるという特徴がある。

【0098】次に、第9の発明について説明する。この発明は第6の発明とほぼ同様の思想であるが、第8の発明に準じ、レジスタファイルの内容転送の代わりにレジスタ写像テーブルの内容を複数回に分割して実現するものである。

【0099】これによって、レジスタ転送のバンド幅を一層低減しながらレジスタ内容の継承が可能になる。なお、レジスタ写像テーブル転送時にレジスタファイルのアクセスが禁止されるサイクルが増加するが、順次レジスタアクセス禁止が解除されるので、その特性に合わせたコードスケジューリングを行い、使用できないレジスタに対するアクセスを遅らせることにより、第6の発明同様、このレイテンシをある程度隠蔽することが可能になる。

【0100】次に、第10の発明について説明する。図24は、この発明の実施例を示す2スレッド並列実行型プロセッサのブロック図である。本発明は、基本的には第8の発明にかかる図20の実施例とほぼ同様であるが、退避用レジスタ写像テーブル98を新たに設け、レジスタフリーテーブル96に対して、退避用レジスタ状態テーブルを付加する。

【0101】空いているスレッド実行手段87が存在しない時にさらにフォーク命令が実行された場合、フォーク命令を実行したプロセッサのフォーク命令デコード時にレジスタ写像テーブル90の内容を退避用レジスタ写像テーブル98にコピーする。このことによって、システム管理ソフトウェアの介入なく、スレッド実行手段8

7の数を超えるスレッド生成要求に対処する。スレッド実行手段87に空きができた時点で、退避用レジスタ写像テーブル98から、空いたスレッド実行手段87のレジスタ写像テーブル90に対してコピーを行い、退避していたスレッドの実行を再開する。これらの管理はスレッド管理手段97によって行われる。

【0102】次に第11の発明について説明する。第11の発明は、1スレッドでフォーク命令によって子スレッドを生成する回数を1回にし、さらに子スレッドは親スレッドが終了するまで終了できないとすることによって得られる効果についての発明である。子スレッドの生成を1回にすることにより、スレッドの生成/消滅は逐次的に行われるようになる。このことを4スレッド並列実行環境に対して適用したモデルが図25である。

【0103】図25に示すように、フォークによって生成される子スレッドの生成先は隣接するスレッド実行手段に限定できる。すなわち、スレッド実行手段#0からは必ずスレッド実行手段#1に対してフォークを行い、スレッド実行手段#1からはスレッド実行手段#2へ、スレッド実行手段#2からはスレッド実行手段#3へ、スレッド実行手段#3からはスレッド実行手段#0へとようにスレッドはリング状にスレッド実行手段に展開される。従って、第4から第10の発明におけるレジスタファイルもしくはレジスタ写像テーブルの内容転送手段を多対多構造からリング状の転送手段に単純化することが可能となる。

【0104】図26は図14の実施例に図25の技術を適用した場合のレジスタファイル回りのブロック図である。図26においてレジスタファイル99は、リング状レジスタファイル転送手段103によって結合している。従って、物理的なレジスタファイル99の位置を工夫することによって、より効率的なハードウェア実装が可能となる。

【0105】次に、本発明のその他の実施の形態について説明する。一つの他の実施の形態としては、第8の発明に対して、第11の発明で用いた子スレッド生成1回限定の特徴を追加し、フォーク時に物理レジスタ継承情報をレジスタフリーテーブルに伝達することなく、論理レジスタ-物理レジスタのリネーミングを実現しつつ、子スレッドへのレジスタ内容継承を実現するものがある。図27にこの実施の形態の実施例として2スレッド並列実行型のプロセッサの構成例を、図28にレジスタフリーテーブル113の詳細図を示す。図27のプロセッサは、基本的には図20のプロセッサと同一構造であるが、レジスタ写像テーブル107からレジスタフリーテーブル113へ、フォーク時に物理レジスタの継承を伝える手段が省略されている。

【0106】また、レジスタフリーテーブル113は、図28に示すように、状態テーブル117の各スレッド実行手段104のエントリが2ビットに拡張されてい

る。このビットの意味を表1に示す。

【0107】

*【表1】

*

| ビット | オーナー権 | 意味 |
|-----|-------|-----------------------------------|
| 00 | × | 当該スレッド実行手段はこの物理レジスタを確保していない |
| 01 | ○ | 当該スレッド実行手段はこの物理レジスタをフォーク後に確保した |
| 10 | × | 当該スレッド実行手段はこの物理レジスタをオーナー権なしでフリーした |
| 11 | ○ | 当該スレッド実行手段はこの物理レジスタをフォーク前に確保した |

【0108】レジスタ確保時には、既にフォークを行ったスレッドかフォークを行っていないスレッドかで状態テーブル117にセットされる値は異なる。従って、フォーク前か後かの情報をスレッド管理手段114から得る必要がある。また、表1中のオーナー権はレジスタを確保したスレッド実行手段104の状態テーブル117に付与されるものであるが、その後のレジスタ継承によって、子スレッド以下に委譲される場合もある。各スレッド実行手段104の状態テーブル117がすべて00の場合、対応するエントリの物理レジスタは未使用状態である。

10※【0109】以下、レジスタ開放論理について説明す

る。この論理はレジスタフリー決定論理115によって決定する。この論理の説明では、Nスレッド同時実行プロセッサモデルとしている。これは、2スレッド同時実行モデルでは、論理が簡単化されてしまうためである。またスレッドは、親スレッド→子スレッド→孫スレッドという順に生成されたものとする。ここで、子スレッドにおいて命令プログラム順序終了時のレジスタ開放時の論理を表2に示す。

【0110】

※20【表2】

| レジスタ 確保時 | 状態遷移前 親 子 孫 | レジスタ 開放時 | 状態遷移後 親 子 孫 | コメント |
|-------------|----------------|-------------|----------------|--------------------|
| フォーク前 | 00 11 00 | フォーク前 | 00 00 00 | レジスタ完全開放 |
| フォーク後 | 00 01 00 | フォーク後 | 00 00 00 | レジスタ完全開放 |
| フォーク前 | 00 11 00 | フォーク後 | 00 00 11 | オーナー権孫スレッドへ委譲 |
| フォーク前 | 00 11 10 | フォーク後 孫開放後 | 00 00 00 | レジスタ完全開放 |
| 親確保 | 11 00 00 | フォーク前 | 11 10 00 | 子スレッド先行開放 |
| 親確保 | 11 00 00 | フォーク後 孫開放前 | 11 00 11 | 子スレッド先行開放/孫2重オーナー権 |
| 親確保 | 11 00 10 | フォーク後 孫開放後 | 00 00 00 | レジスタ完全開放 |

【0111】この表2から明らかなように、孫スレッドへのレジスタ継承が行われている際には、親フィールド、子フィールド、孫フィールドの3種類のフィールド値と開放しようとしているスレッドのフォーク状態によって状態遷移が決定される。これ以外の状態遷移は通常ではあり得ずエラーとなる。

【0112】この論理を用いることによって、フォーク命令実行時にレジスタフリーテーブル113は多数の状態テーブル117の内容を変更することなく、レジスタの継承/開放が可能になる。

【0113】さらに他の実施の形態では、図29に示したように、レジスタ使用中ビット121を付加する。このレジスタ使用中ビットは、物理レジスタを確保する際に、フリーレジスタ検索論理119によって未使用のものを見つけ、使用中にセットする。このことによって状態テーブル120のすべてのエントリが00もしくは未使用であるという状態のレジスタを検索する論理が簡単化される。レジスタ使用中ビット121を使用中状態か

ら、未使用状態にセットするのは、レジスタフリー決定論理118が、状態テーブル120の内容を書き換える際に同時に行う。

【0114】

【発明の効果】以上説明したように、本発明によれば、スレッドを並列に処理する際に、親スレッドから子スレッドに対して共有メモリを介せずにレジスタ内容の継承が可能になり、スレッド生成時のオーバーヘッドを減らすことができる。また、このレジスタ内容の継承をout-of-order実行を行うプロセッサに対しても、フォーク命令前後間においても実現したため、スレッド生成にともなうオーバーヘッドを減らすことが可能になり、粒度の細かいスレッドに対してもスレッドレベル並列処理による処理速度の向上が実現できる。

【図面の簡単な説明】

【図1】第1の発明におけるレジスタ内容の継承方法の概念図である。

【図2】第1の発明におけるレジスタ内容の継承方法を

実現する2スレッド並列実行型プロセッサの実施例のブロック図である。

【図3】図2に示した実施例のプロセッサにおけるスレッド生成時のタイミングチャートである。

【図4】第2の発明を適用した2スレッド並列実行型プロセッサの実施例のブロック図である。

【図5】レジスタ選択手段#1 12bの構成例を示すブロック図である。

【図6】レジスタファイル#0 13aの構成例を示すブロック図である。

【図7】図4に示した実施例のプロセッサのタイミングチャートである。

【図8】第3の発明の動作説明図である。

【図9】第4の発明を適用した2スレッド並列実行型プロセッサの実施例のブロック図である。

【図10】命令キュー#0 27aの構成例を示すブロック図である。

【図11】リオーダバッファ#0 24aの構成例を示すブロック図である。

【図12】レジスタデータセクタ装置#1 26bの処理の論理を示すフローである。

【図13】図9のプロセッサにおけるフォーク時の動作モデルを示す図である。

【図14】第4の発明を適用した4スレッド並列実行2演算ユニット型プロセッサの実施例のブロック図である。

【図15】第5の発明を適用した2スレッド並列実行型プロセッサの実施例のブロック図である。

【図16】図15のプロセッサにおけるフォーク時の動作モデルを示す図である。

【図17】第6の発明の実施例のブロック図である。

【図18】図17の実施例のタイミングチャートである。

【図19】第7の発明の実施例のブロック図である。

【図20】第8の発明を適用した2スレッド並列実行型プロセッサの実施例のブロック図である。

【図21】レジスタ写像テーブル76aの詳細なブロック図である。

【図22】レジスタフリーテーブル82の詳細なブロック図である。

【図23】レジスタビジューテーブル81の詳細なブロック図である。

【図24】第10の発明を適用した2スレッド並列実行型プロセッサの実施例のブロック図である。

【図25】第11の発明の実施例のモデルを示す図である。

【図26】図14の実施例に図25の技術を適用した場合のレジスタファイル回りのブロック図である。

【図27】本発明の他の実施の形態の実施例である2スレッド並列実行型のプロセッサのブロック図である。

【図28】レジスタフリーテーブル113の詳細図である。

【図29】本発明の他の実施の形態におけるレジスタフリーテーブルの詳細図である。

【図30】従来のリオーダバッファ方式によるレジスタリネーミング機構の構成を示すブロック図である。

【図31】従来のレジスタ写像テーブル方式によるレジスタリネーミング機構の構成を示すブロック図である。

【図32】従来のマルチスレッド型のプロセッサの構成を示すブロック図である。

【図33】Multiscalar Processorの構成を示すブロック図である。

【符号の説明】

1…親スレッド（新スレッドを生成するスレッド）

2…スレッド生成命令（フォーク命令）

3…子スレッド（新スレッド）

4…スレッド管理手段

5a, 5b…スレッド実行手段

6a, 6b…レジスタファイル

7…レジスタ内容一括転送手段

8…共有メモリ

9a, 9b…スレッド実行手段

10a, 10b…命令キャッシュ

11a, 11b…命令デコーダ

12a, 12b…レジスタ選択手段

13a, 13b…レジスタファイル

14a, 14b…演算ユニット

15…スレッド管理手段

16…レジスタ選択テーブル

17…レジスタ選択ビット

18…レジスタ内容メモリ

19…転送終了ビット

20…更新ビット

21a, 21b…スレッド実行手段

22a, 22b…命令キャッシュ

23a, 23b…命令デコーダ

24a, 24b…リオーダバッファ

25a, 25b…レジスタファイル

26a, 26b…レジスタデータセクタ装置

27a, 27b…命令キュー

28a, 28b…演算ユニット

29…スレッド管理手段

30…キューエントリ格納論理

31…発行命令決定論理

32…命令キューエントリ

33…エントリ有効ビット

34…レジスタ内容／リオーダバッファタグ格納メモリ

35…レジスタ内容有効ビット

36…結果書き込み場所指定タグ

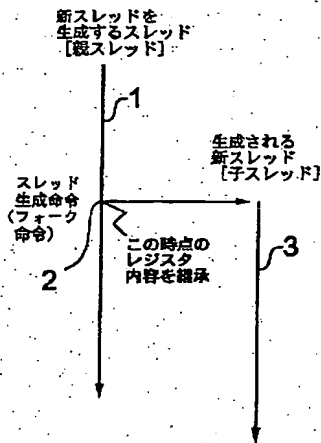
37…命令コード格納メモリ

38…レジスタデータ供給決定論理
 39…エントリシフト制御論理
 40…リオーダーバッファエントリ
 41…エントリ有効ビット
 42…結果書き込み指定タグ
 43…レジスタ番号
 44…スレッド生成ビット
 45…演算データ格納メモリ有効ビット
 46…演算データ格納メモリ
 47a, 47b…結果振り分け装置
 48a, 48b…レジスタデータ選択装置
 49a, 49b…命令キュー
 50a, 50b…スレッド実行手段
 51a, 51b…命令キャッシュ
 52a, 52b…命令デコーダ
 53a, 53b…リオーダーバッファ
 54a, 54b…レジスタファイル
 55a, 55b…レジスタデータセクタ装置
 56a, 56b…命令キュー
 57a, 57b…演算ユニット
 58…スレッド管理手段
 59a, 59b…レジスタファイル
 60…マルチプレグサ
 61…デバイダ
 62a, 62b…参照許可ビット
 63a, 63b…スレッド実行手段
 64a, 64b…命令キャッシュ
 65a, 65b…命令デコーダ
 66a, 66b…リオーダーバッファ
 67a, 67b…レジスタファイル
 68a, 68b…レジスタデータセクタ装置
 69a, 69b…命令キュー
 70a, 70b…演算ユニット
 71…スレッド管理手段
 72…退避用レジスタファイル
 73a, 73b…スレッド実行手段
 74a, 74b…命令キャッシュ
 75a, 75b…命令デコーダ
 76a, 76b…レジスタ写像テーブル
 77a, 77b…命令キュー
 78…物理レジスタファイル
 79a, 79b…演算ユニット
 80a, 80b…有効命令順序バッファ
 81…レジスタビジューテーブル
 82…レジスタフリーテーブル
 83…スレッド管理手段
 84…レジスタフリー決定論理
 85…フリーレジスタ検索論理
 86…状態テーブル
 87a, 87b…スレッド実行手段

88a, 88b…命令キャッシュ
 89a, 89b…命令デコーダ
 90a, 90b…レジスタ写像テーブル
 91a, 91b…命令キュー
 92…物理レジスタファイル
 93a, 93b…演算ユニット
 94a, 94b…有効命令順序バッファ
 95…レジスタビジューテーブル
 96…レジスタフリーテーブル
 10 97…スレッド管理手段
 98…退避用レジスタ写像テーブル
 99a, 99b, 99c, 99d…レジスタファイル
 100a, 100b…結果振り分け装置
 101a, 101b…レジスタデータ選択装置
 102a, 102b, 102c, 102d…リオーダーバッファ
 103…リング状レジスタファイル転送手段
 104a, 104b…スレッド実行手段
 105a, 105b…命令キャッシュ
 20 106a, 106b…命令デコーダ
 107a, 107b…レジスタ写像テーブル
 108a, 108b…命令キュー
 109…物理レジスタファイル
 110a, 110b…演算ユニット
 111a, 111b…有効命令順序バッファ
 112…レジスタビジューテーブル
 113…レジスタフリーテーブル
 114…スレッド管理手段
 115…レジスタフリー決定論理
 30 116…フリーレジスタ検索論理
 117…状態テーブル
 118…レジスタフリー決定論理
 119…フリーレジスタ検索論理
 120…状態テーブル
 121…レジスタ使用中ビット
 122…レジスタファイル
 123…リオーダー・バッファ
 124…レジスタ写像テーブル
 125…レジスタファイル
 40 126…有効命令順序バッファ
 127…レジスタビジューテーブル
 128…レジスタフリーテーブル
 129…命令取得装置
 130…命令解読装置
 131…機能実行装置
 132…命令依存解析装置
 133…命令調停装置
 134…シーケンサ
 135…プロセッシングユニット
 50 136…結合ネットワーク

137...データバンク
138...命令キャッシュ
139...実行ユニット
140...レジスタファイル

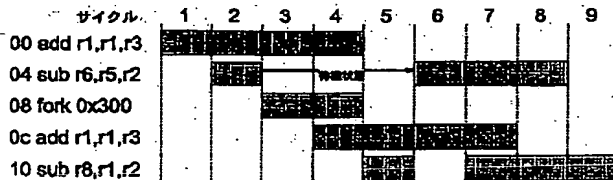
【図1】



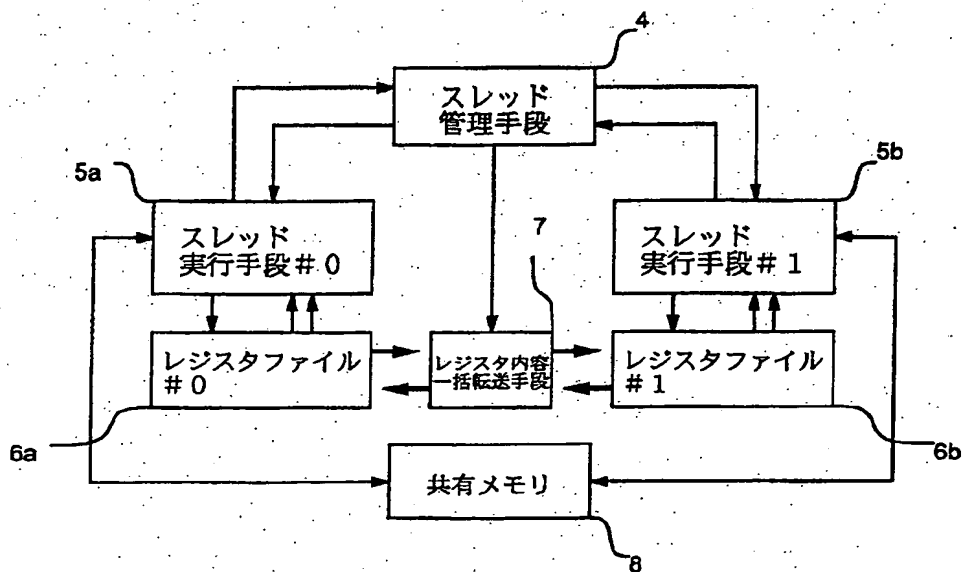
141...ARB (Address Resolution Buffer)
142...データキャッシュ
150...レジスタ内容転送手段

【図8】

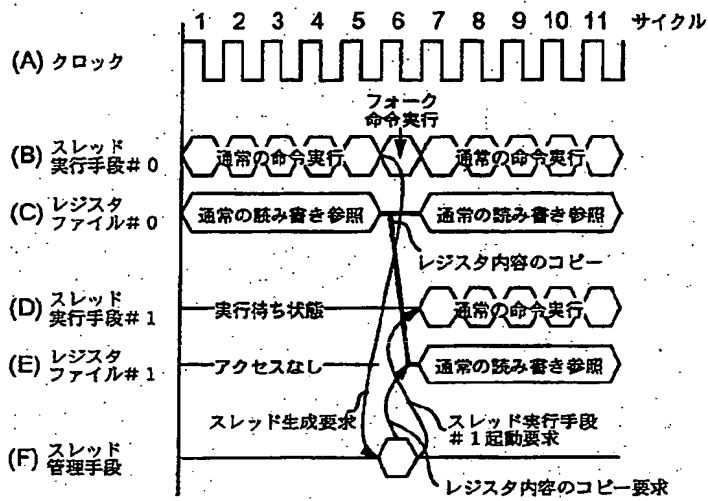
out-of-order 実行順序例



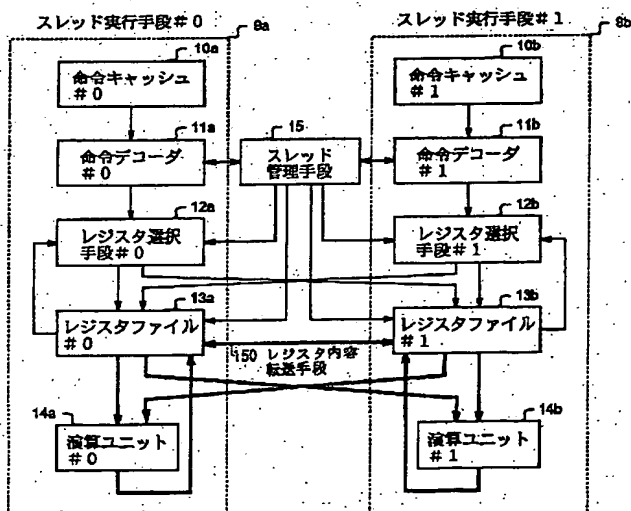
【図2】



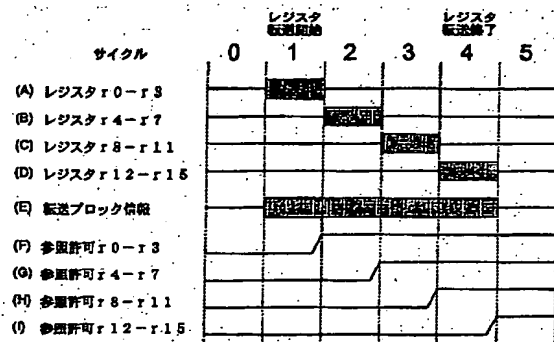
【図3】



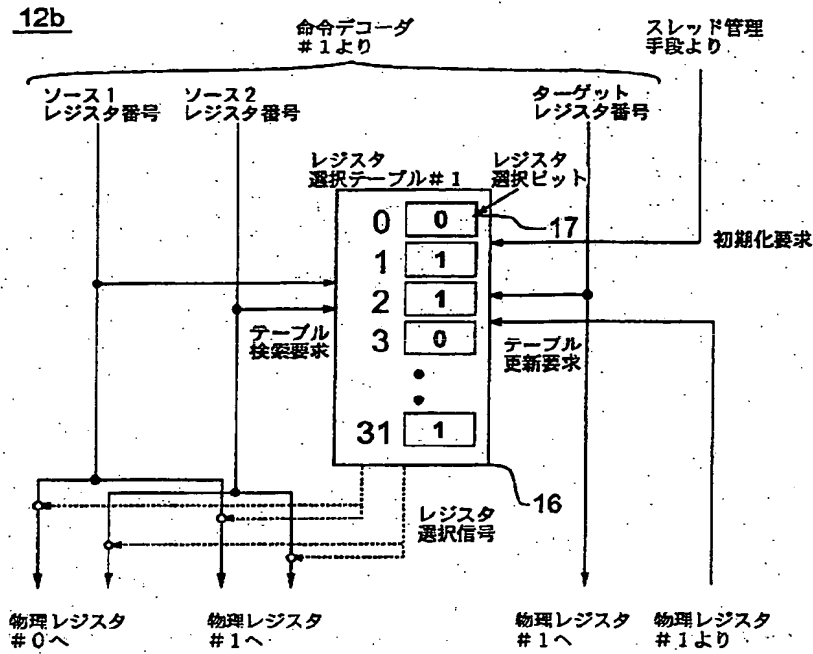
【図4】



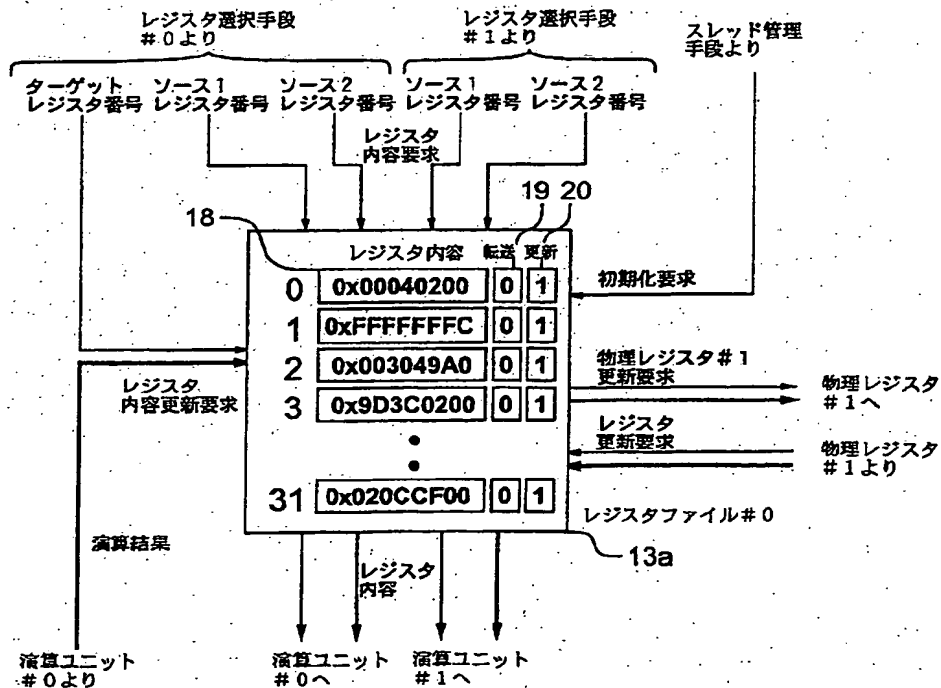
【図18】



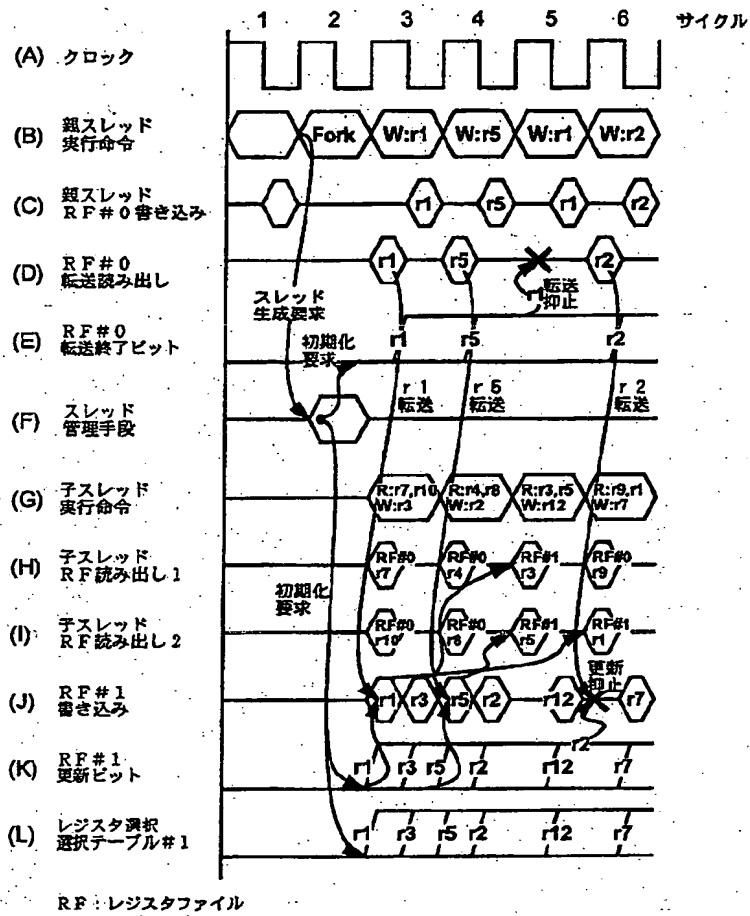
【図5】



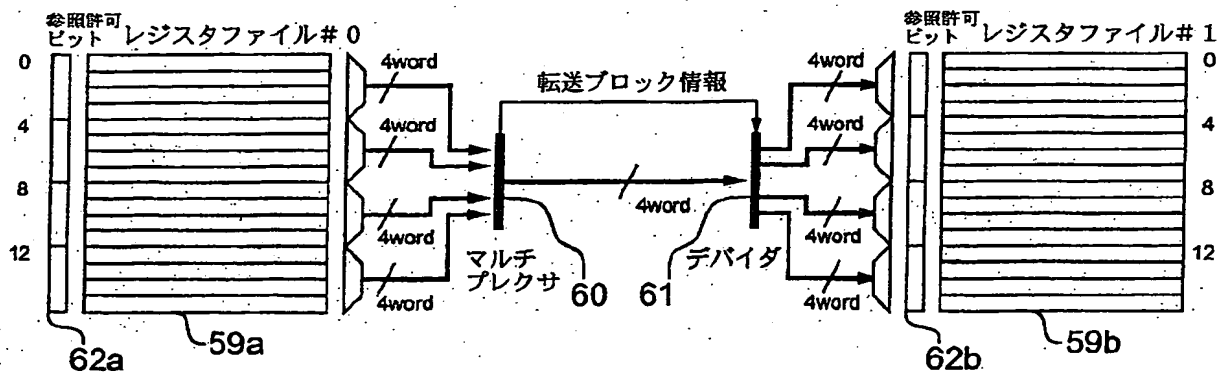
【図6】



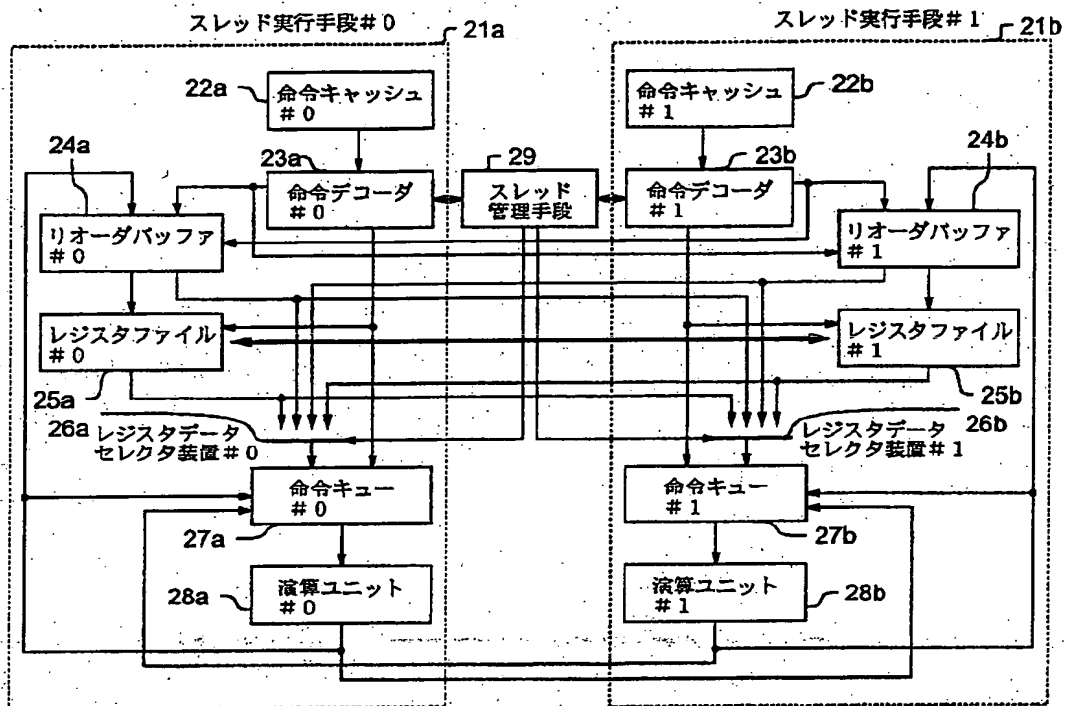
【図7】



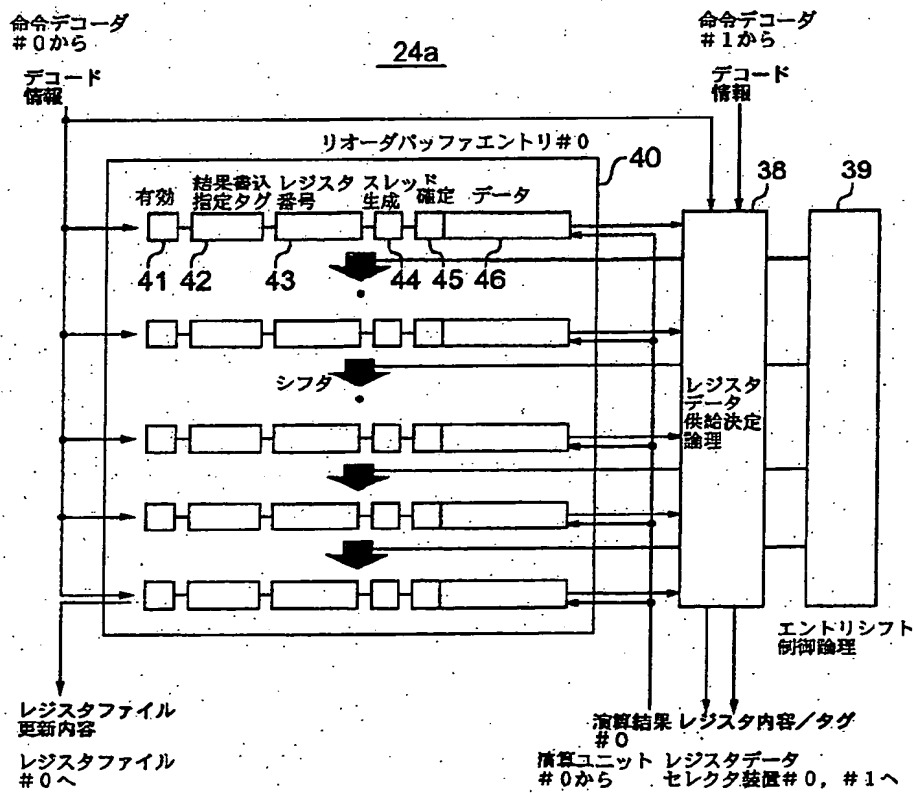
【図17】



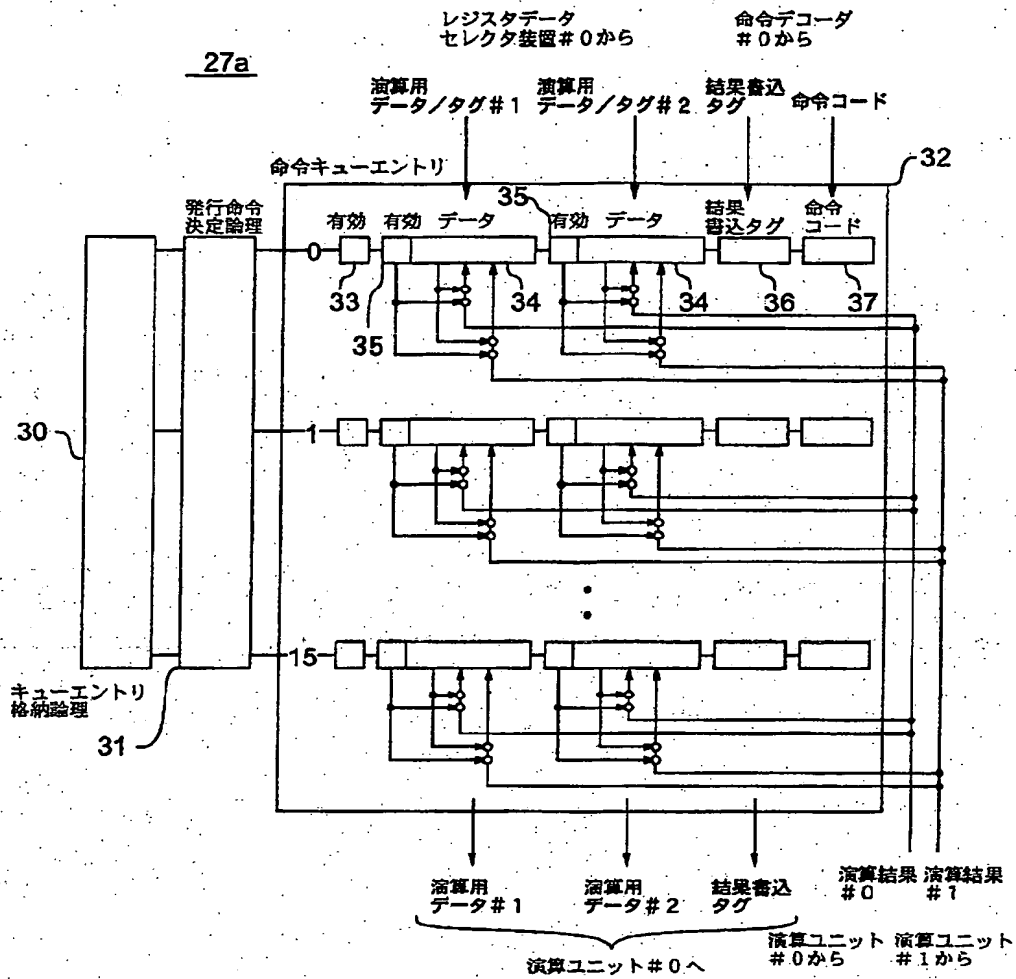
【図9】



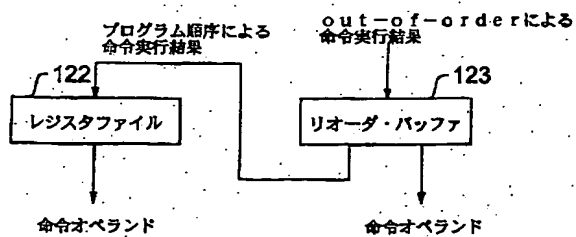
【図11】



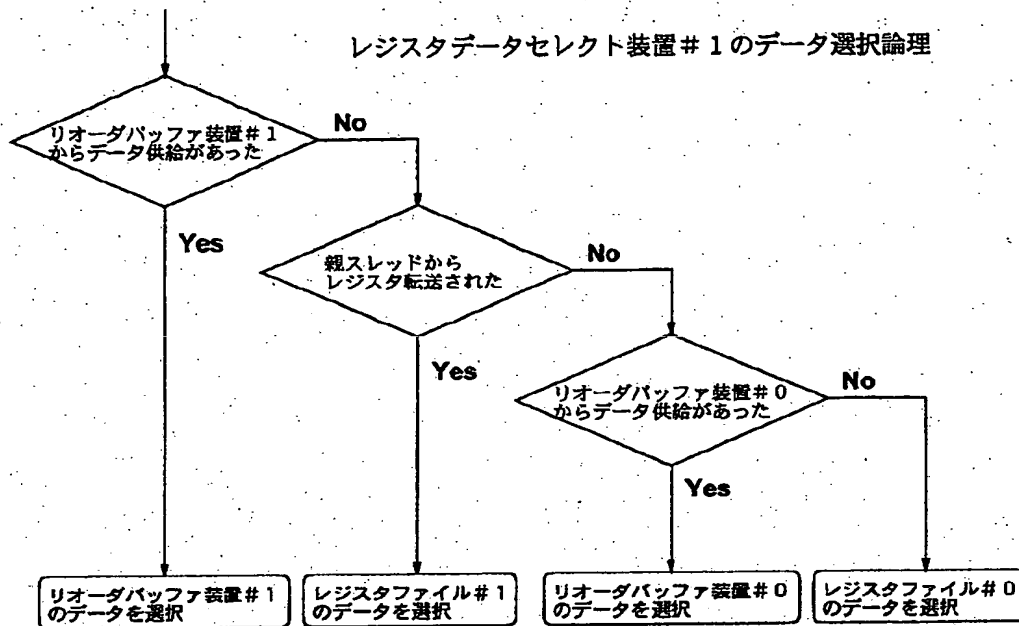
【図10】



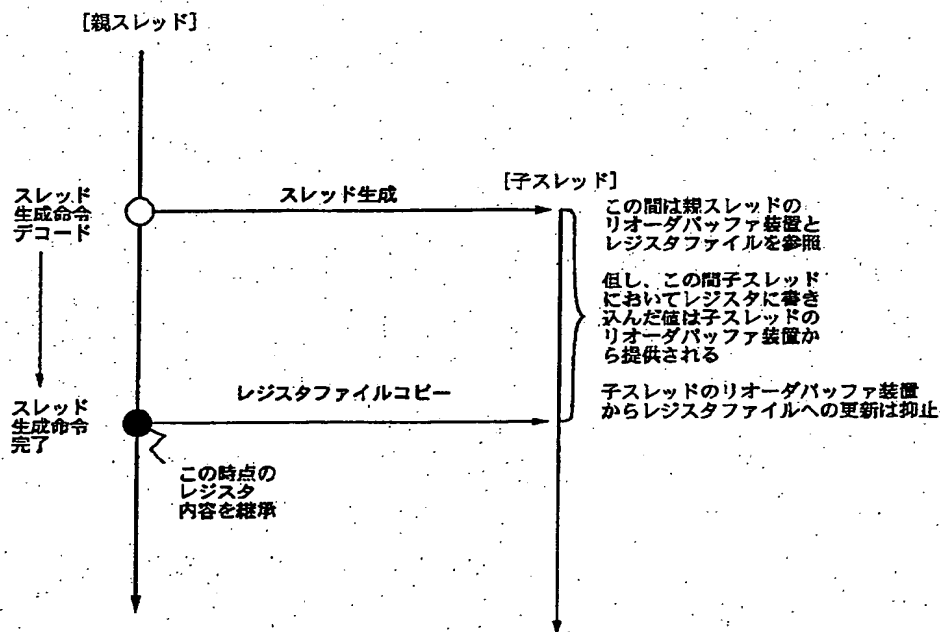
【図30】



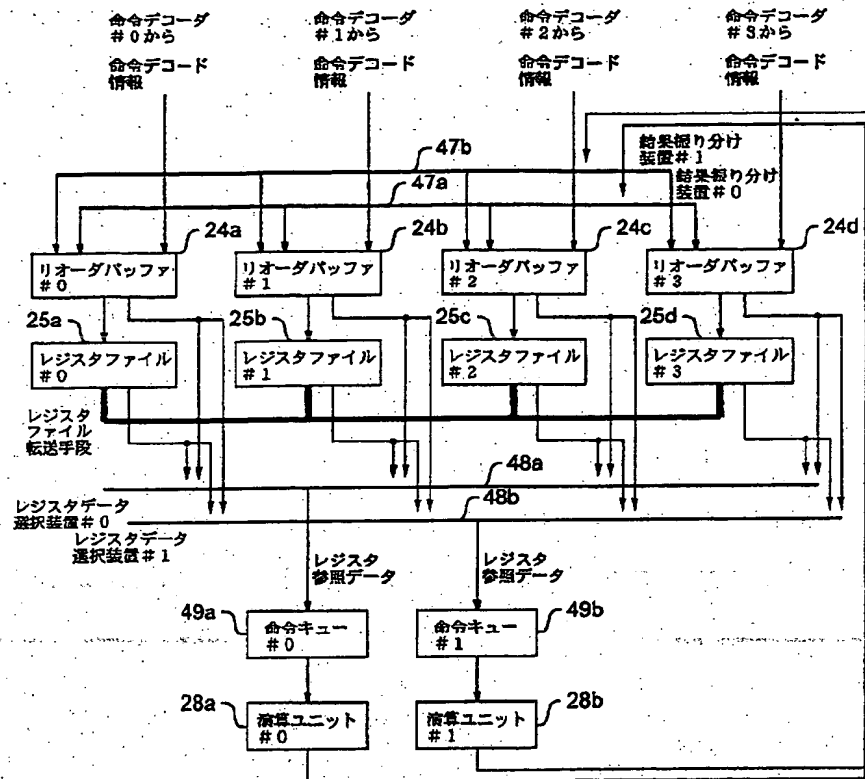
【図12】



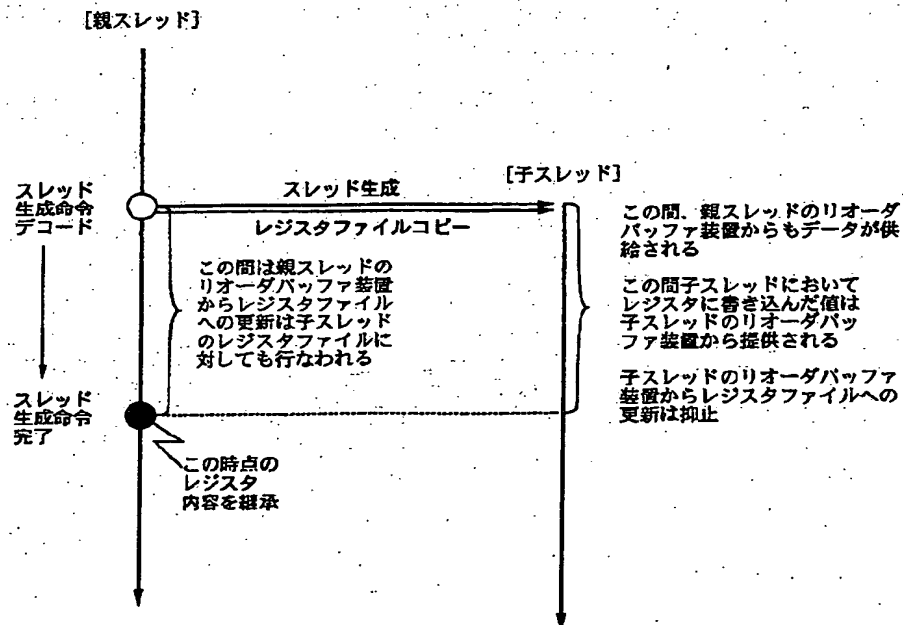
【図13】



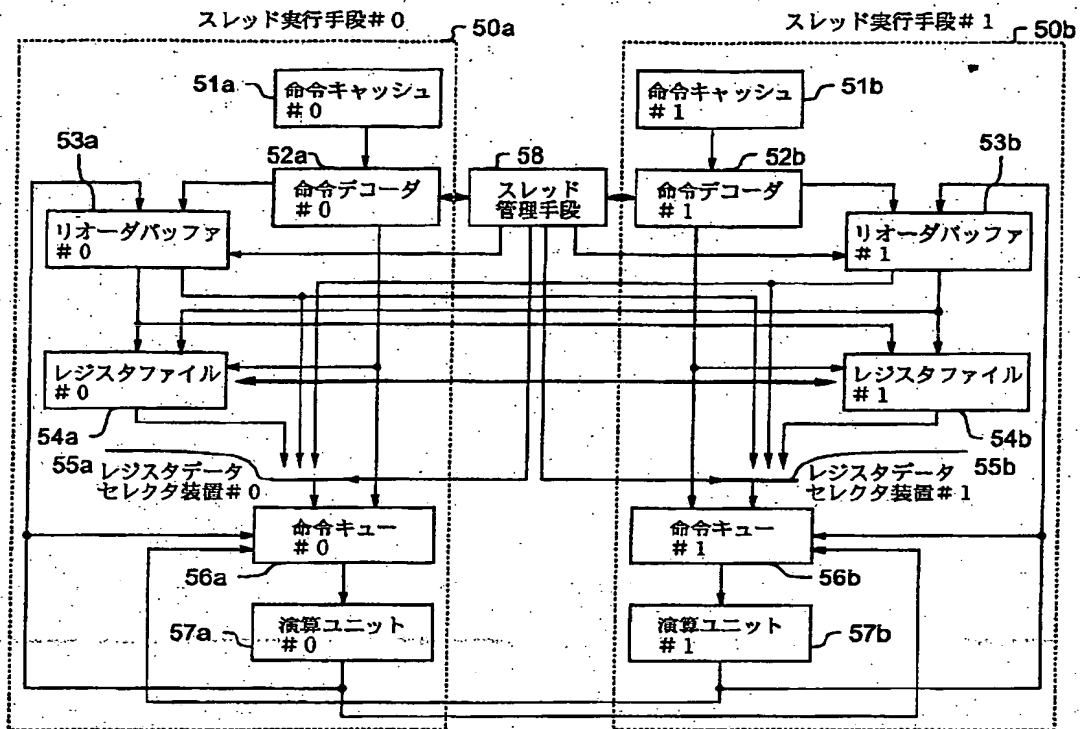
【図14】



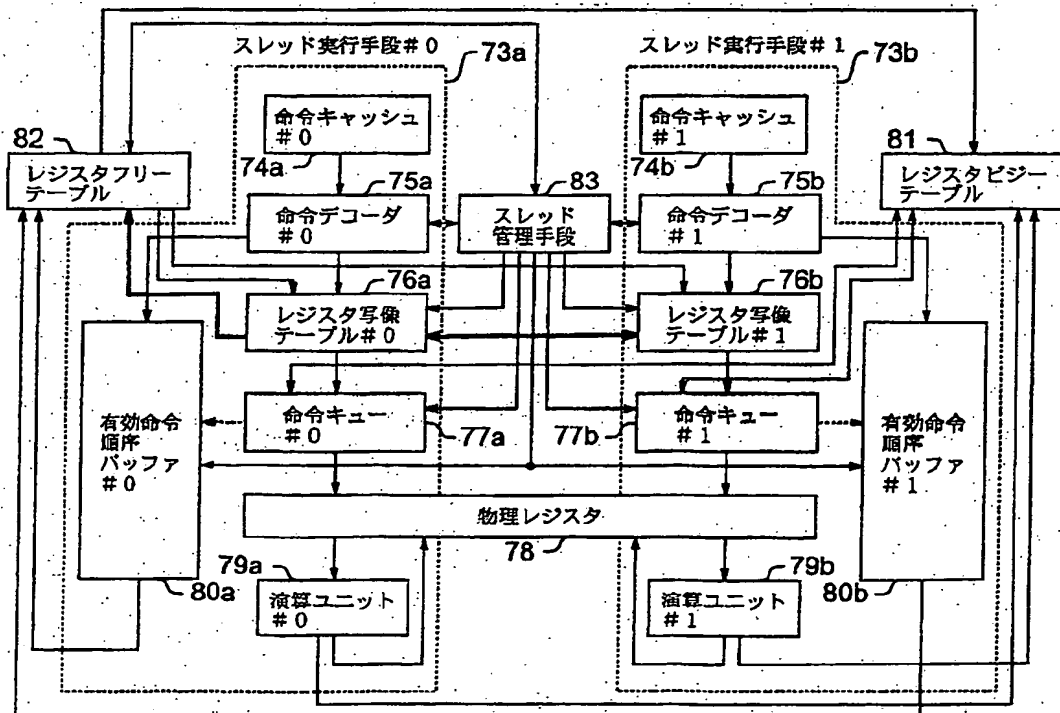
【図16】



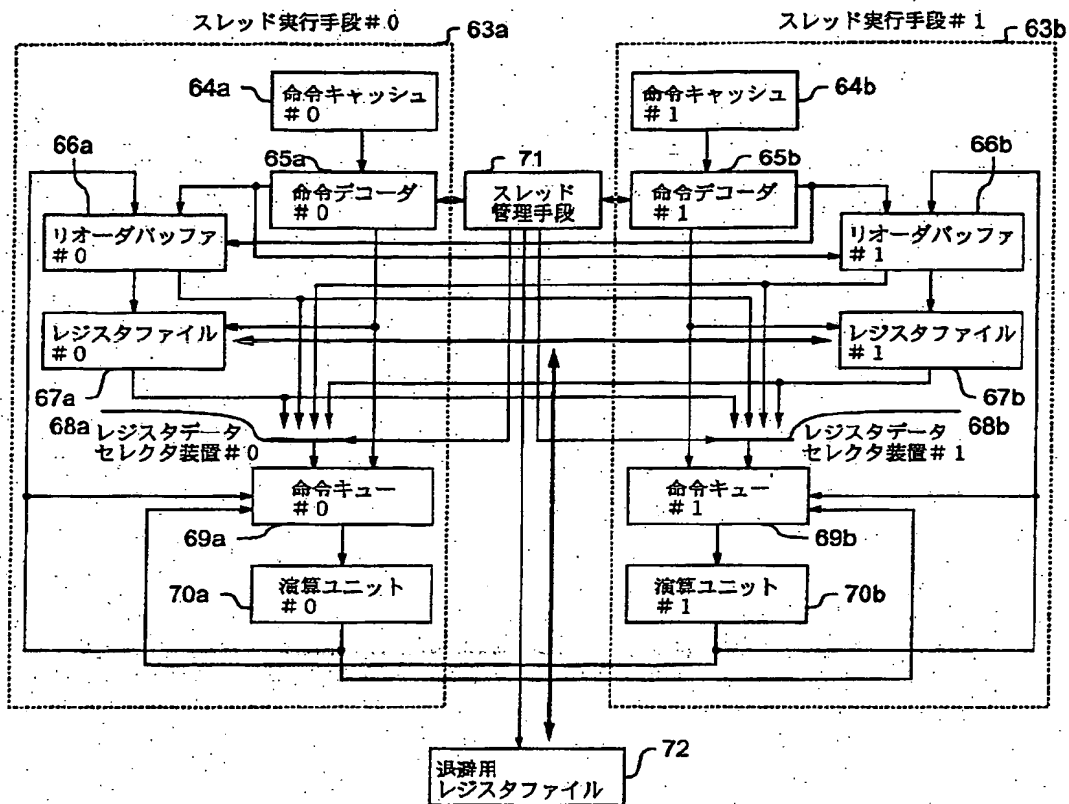
【図15】



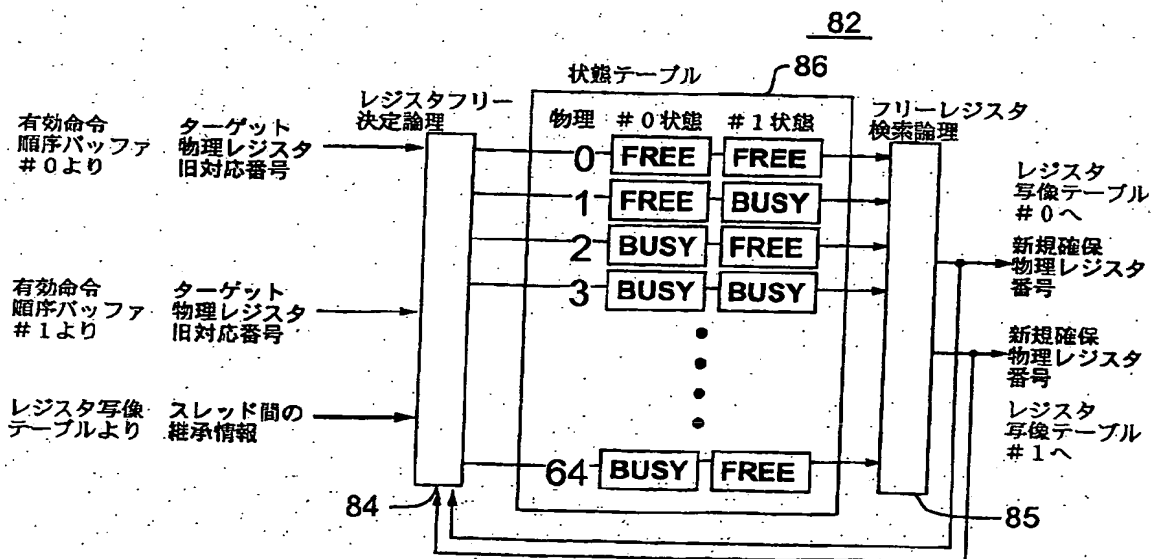
【図20】



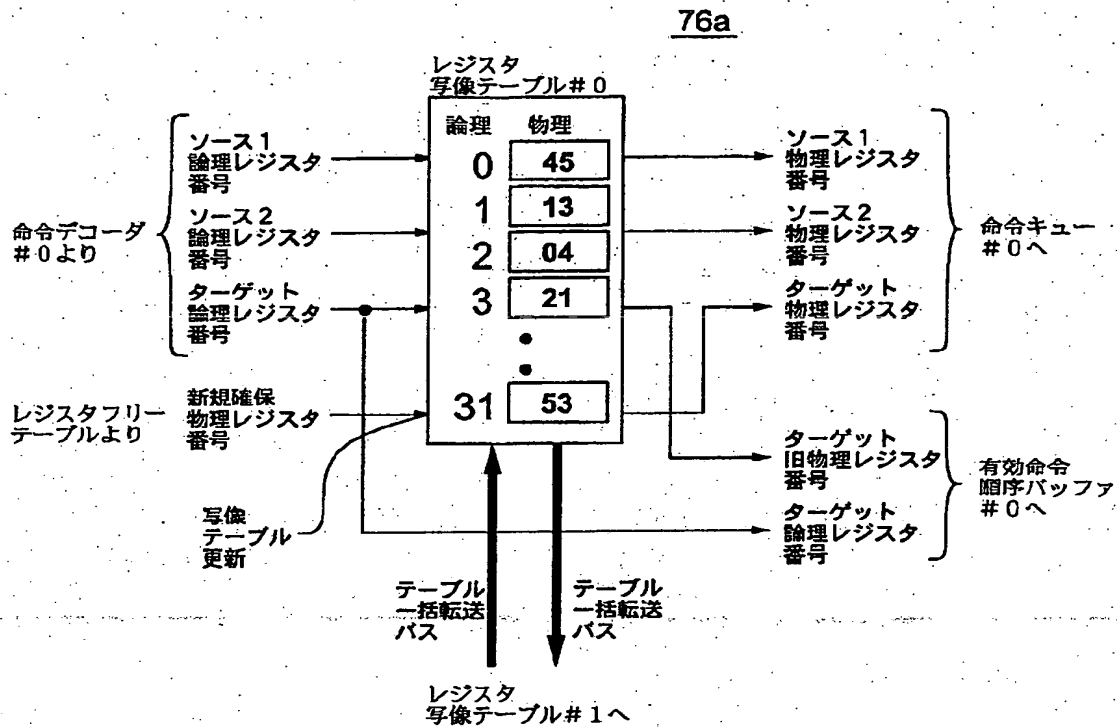
【図19】



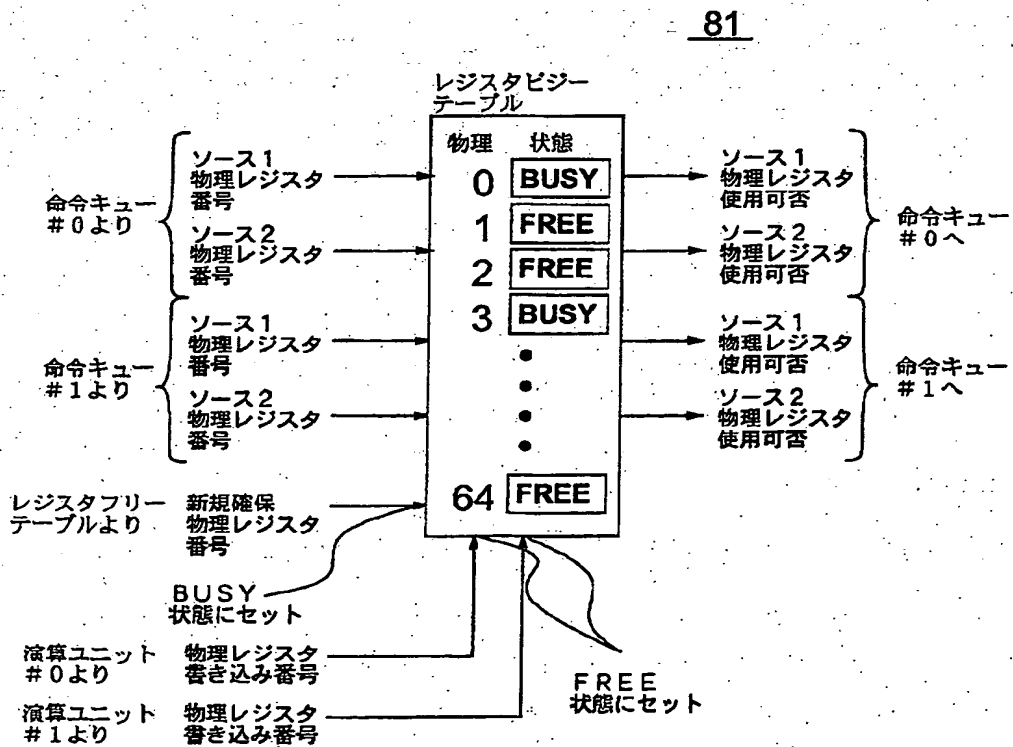
【図22】



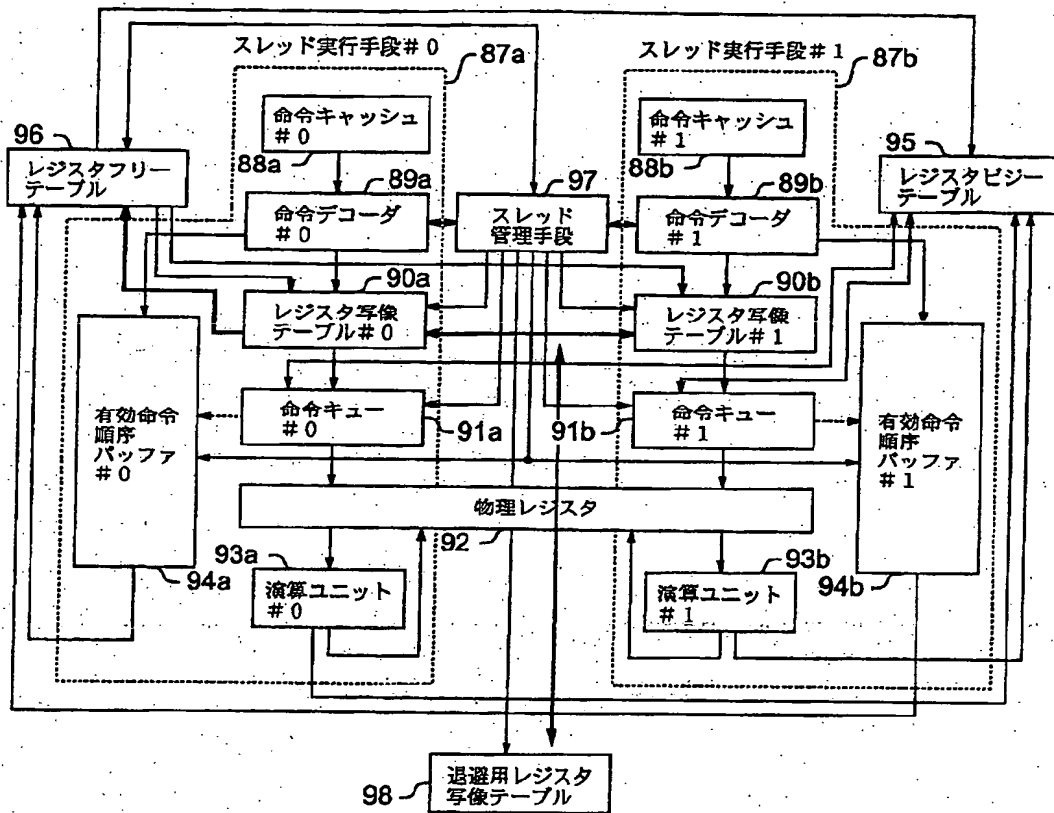
【図21】



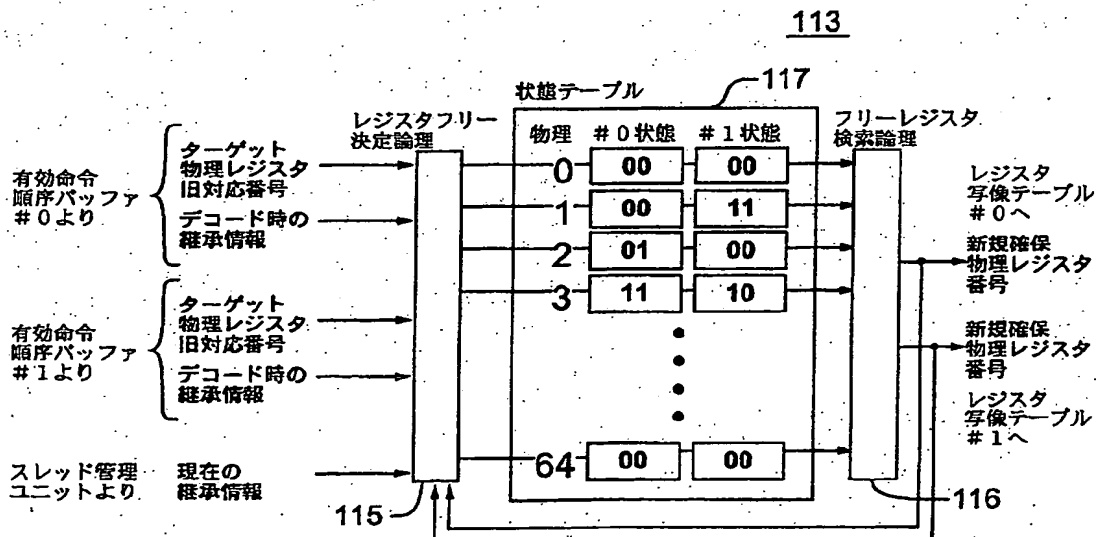
【図23】



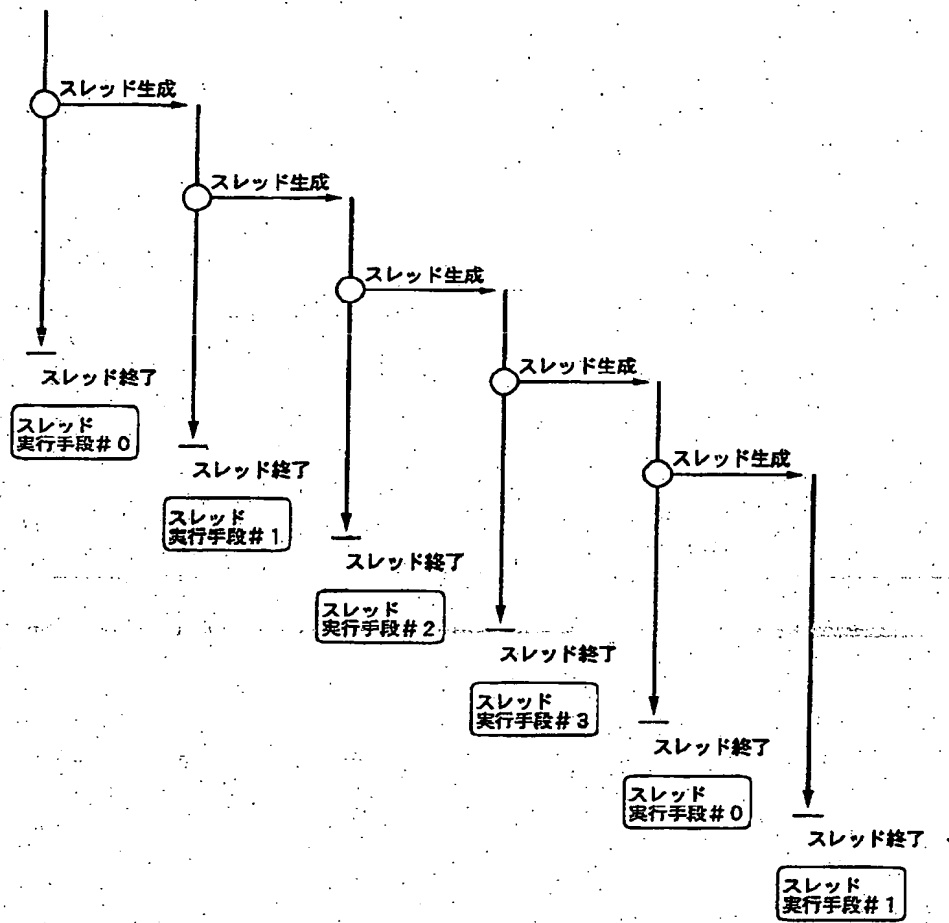
【図24】



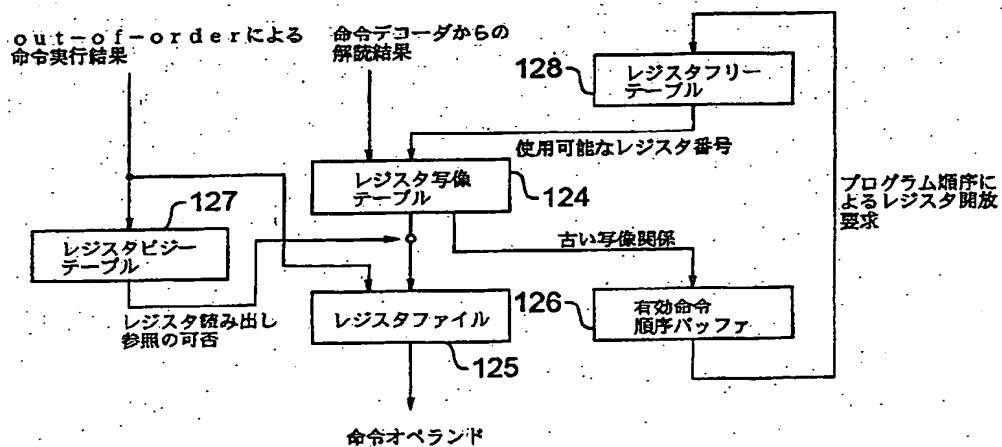
【図28】



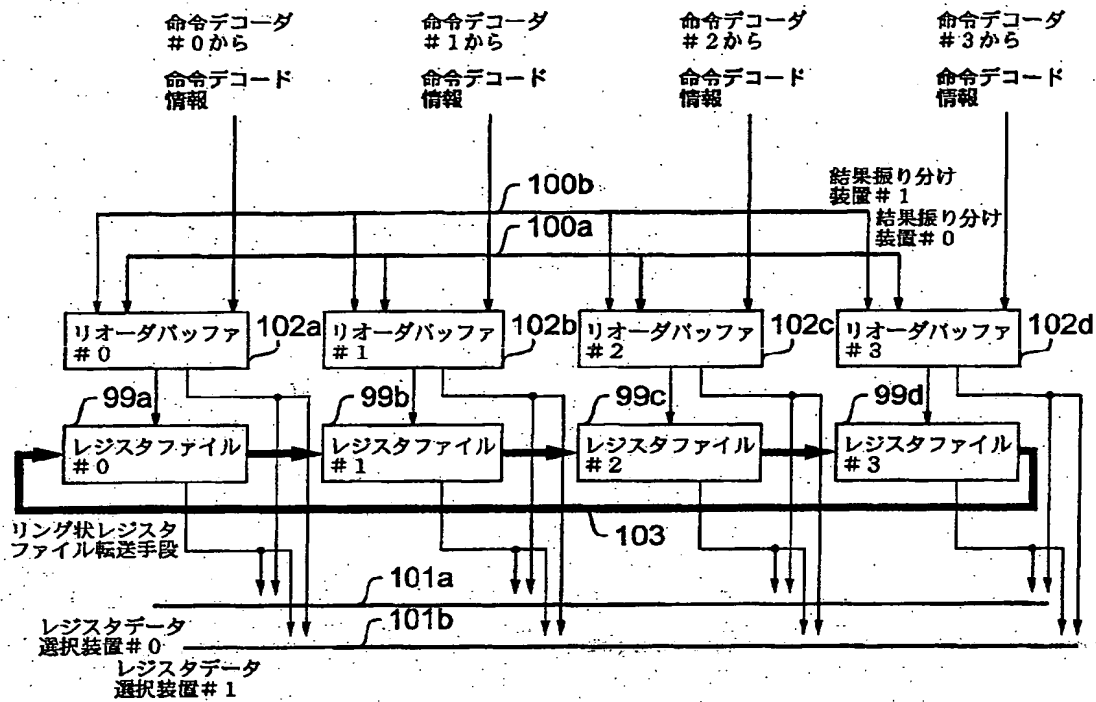
【図25】



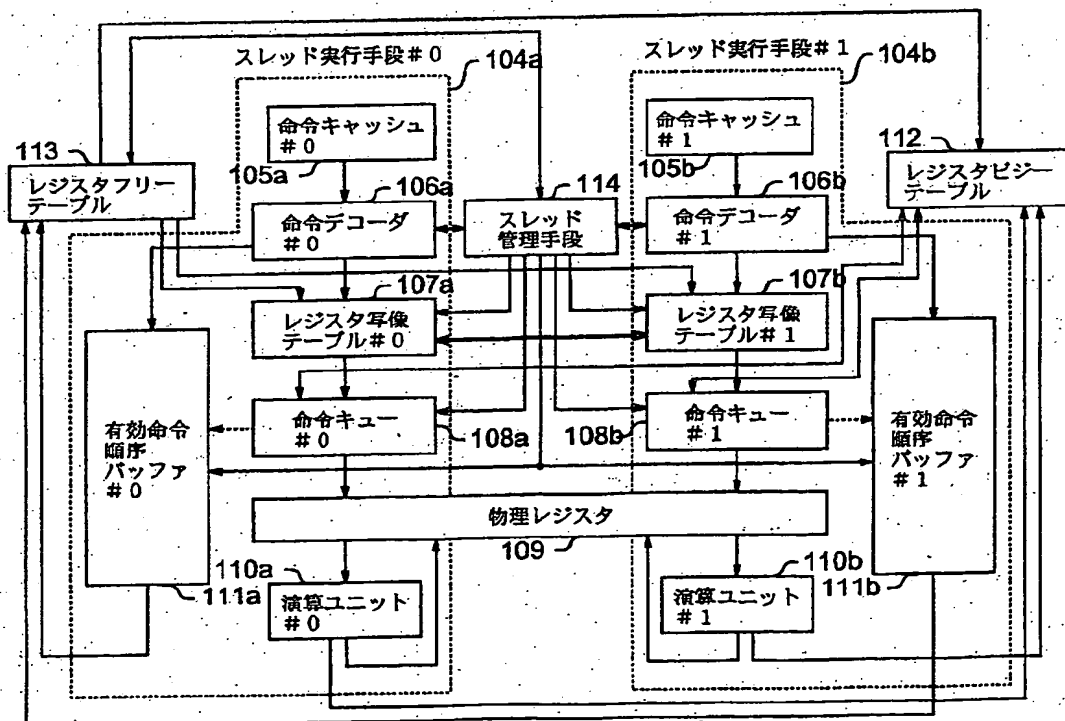
【図31】



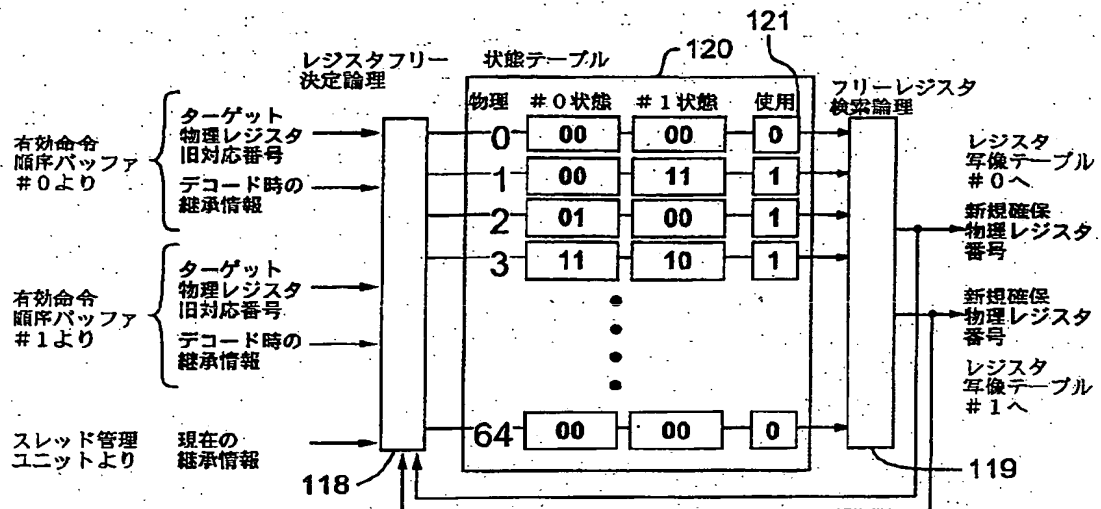
【図26】



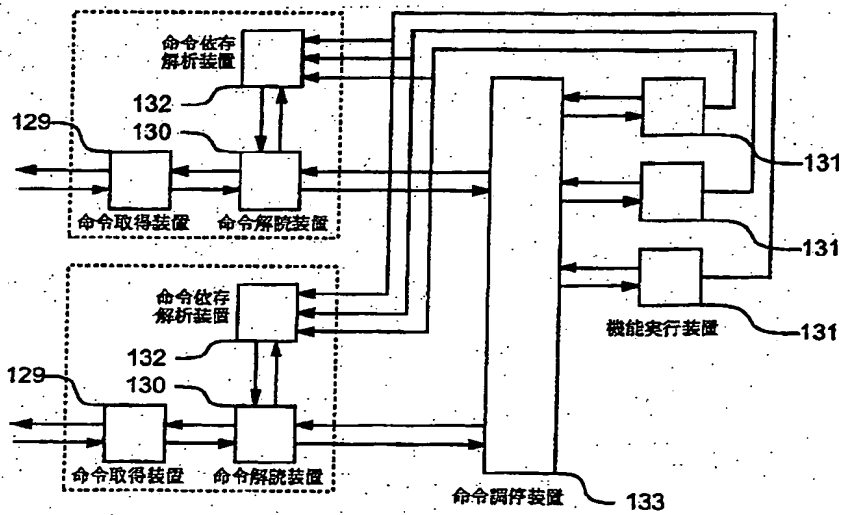
【図27】



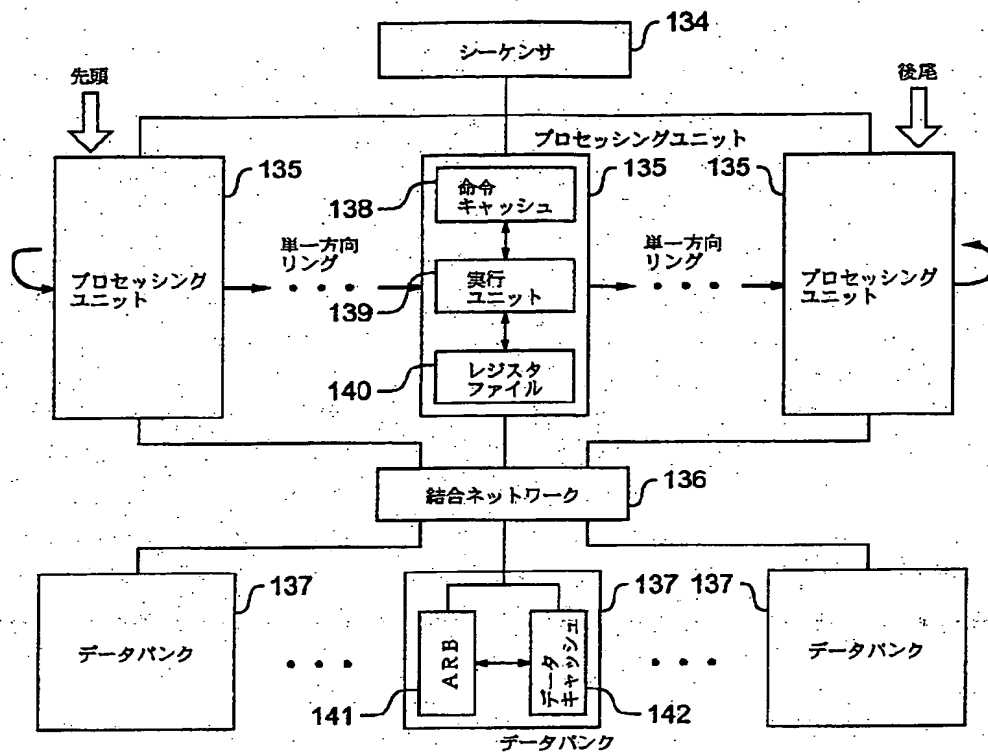
【図29】



【図32】



【図33】



**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☒ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER: _____**

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.